活学活用——

Linux 教程

编者 李洛 黄达峰

目　　录

第 1 章　Linux 入门

本章提要

- Linux 的背景
- 安装 Linux 系统
- 使用 Linux 系统
- 配置 Linux 系统

1.1　Linux 的背景

1.1.1　Unix

要介绍 Linux，必须提及 Unix。

Unix 操作系统的发展历史悠久，起源于 1969 年 AT&T 的贝尔实验室，作者是 Ken Thompson。在 1972 年 Dennis Ritchie 创立 C 语言后，他们两人又合力用 C 语言重写了 Unix 操作系统，大幅增加其可移植性，然后 Unix 操作系统开始蓬勃发展（如图 1-1）。

在 Unix 发展的早期，任何感兴趣的机构或个人只需要向贝尔实验室支付一笔数目极小的名义上的费用就可以完全获得 Unix 的使用权，并包含源代码和使用帮助手册。这些使用者主要是一些大学和科研机构，他们对 Unix 的源代码进行扩展和定制，以适合各自的需要。

自 20 世纪 70 年代后期起，加州大学伯克利分校（UC，Berkeley）的计算机科学家们致力于 Unix 的研究，改良和发展 Unix。这些改进包括现在大名鼎鼎的 TCP/IP 协议的创立在内。他们的努力促成了一个重要的 Unix 分支——BSD(Berkeley Systems Distribution) Unix 系列，例如 FreeBSD、NetBSD 和 OpenBSD 等版本的 Unix，到目前为止这些版本还在活跃发展中。

Unix 操作系统被设计用来让许多的程序员能够同时访问一台计算机并共享使用其上的资源，它通过使用分时(Time Sharing)技术让每个同时访问的用户都觉得只有他自己一个人在使用该计算机。Unix 具有众多的优秀特性：多任务、多用户、可移植性、大量的实用程序和应用程序等，因此三十多年来长盛不衰。

【注】 大写的“UNIX”是 AT&T 的商标（已经转让给其他机构），专指 AT&T 的 Unix 操作系统。如今，当人们谈论“Unix”的时候它所指的已经不是那个起源的 “Unix”了，而通常是指一个“类 UNIX”的操作系统，广义上也包括 Linux。现在有很多个分支或版本的 Unix，而 Linux 已成为其中最流行的一个。

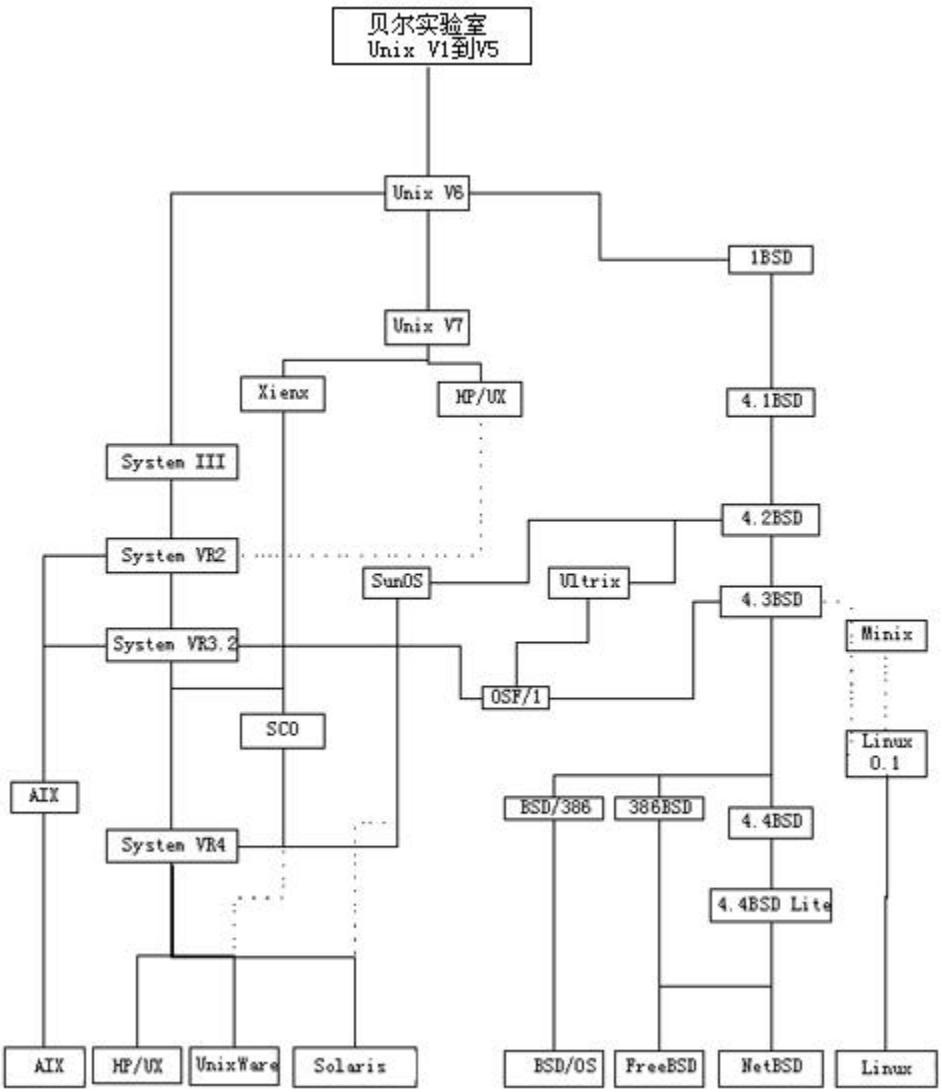

图 1-1 Unix 家族简图

1.1.2 自由软件

Unix 不断发展，逐渐演变成多个分支，形成各种版本。商业化是其中的一个主要发展方向（如图 1-1 所示的“AIX”、“HP/UX”、“UnixWare”和“Solaris”四个分支），因此 Unix 的使用受到各种各样的限制。许多厂商购买了 Unix 的源代码，开发自己的 Unix 版本。这些版本一般是专用的，而且针对各自生产的计算机硬件。用户购买这些专有的 Unix 系统（及其硬件），需要支付昂贵的使用费用。如果需要一份源代码的拷贝，必须额外支付费用；由于版权问题，用户对这些源代码的改进只能自己享用，不可以与他人分享。

在 1984 年，Richard Stallman 觉察到了这些变化。他辞去麻省理工学院(MIT)的工作，建立了一个 GNU(GNU's Not UNIX) 计划，准备开发一套完整的称为 GNU 的属于自由软件的“类

UNIX”操作系统。

Richard Stallman 提出“自由软件”(Free Software)的理念，希望给予用户充分的使用软件的自由。自由软件不一定是免费的(Gratis)，它所表达的是用户可自由执行程序、研究源代码、做修改，甚至重新发布，至于要免费或收费则并无限制。GNU 计划对“自由软件”作出定义：

对于某个软件，如果使用者:

I. 不论目的为何，有使用该软件的自由。

II. 有研究该软件如何运作的自由，并且可以改写该软件来符合使用者自身的需求。(取得该软件之源码为达成此目的之前提。)

III. 有重新发布该软件的自由，不管免费或者收费。

IV. 有改善再利用该软件的自由，并且可以发表改写版供公众使用，如此一来，整个社群都可以受惠。

则该软件就可以被称之为自由软件。

在过去的 20 年中，GNU 计划催生出数量众多的在计算机领域影响巨大的自由软件。这些自由软件广泛使用“GNU GPL”(GNU General Public License，GNU 通用公共许可证)来保护每个使用者都享有这些软件自由。这通常称为“copyleft”。

【注】 有两个术语与“自由软件”容易混淆。一个是“Freeware”，指“免费软件”，其关注的重点是价格问题。另一个是“Shareware”，指“共享软件”，是以“先使用后付费”的方式销售的享有版权的软件。根据共享软件作者的授权，用户可以从各种渠道免费得到它的拷贝，也可以自由传播它。用户总是可以先使用或试用共享软件，认为满意后再向作者付费；如果用户认为它不值得花钱购买，则可以停止使用。

1.1.3 开源软件

开源软件，即开放源代码软件(Open Source Software)是在开放源代码许可证(Open-source License)下发布的软件，以保障软件用户自由使用及接触源代码的权利。这同时也保障了用户自行修改、复制以及再分发的权利。

开源，即开放源代码(Open Source)，不仅仅指开放源代码软件，它同时也是一种软件开发模式的名称。开放源代码开发模式的名字及其特点最早是由 Eric Raymond 在他的著作《大教堂与市集》(The Cathedral and the Bazaar)等一系列论文集中提出并探讨的。

严格地来说，“开源软件”与“自由软件”是两个不同的概念，只要符合开源软件定义的软件就能被称为开源软件。自由软件是一个比开源软件更严格的概念，因此所有自由软件都是开放源代码的，但不是所有的开源软件都能被称为“自由”。但在现实上，绝大多数开源软件也都符合自由软件的定义。比如，遵守 GPL 许可证和 BSD 许可证的软件都是开放的并且是自由的。

开源软件运动是一个主要由软件工程师及其他计算机用户参与的声势浩大的运动。它是自由软件运动的一个分支，但两者的差别并不明显。一般而言，自由软件运动是基于政治及哲学思想（有时被称为所谓的黑客文化）的理想主义运动，而开放源代码运动则主要注重软件本身的质量提升。

1.1.4 Linux 内核

在 Unix 发展的早期，大概从 1974 年起，Unix 逐渐广泛用于大学教学。但后来，由于版权问题，Unix 的源代码不再适用于教学。

1987 年 Andy Tanenbaum 编写了 Minix 操作系统作为教学的工具。Minix 的意思为 mini-UNIX，它是一个简化了的“类 UNIX”操作系统，适合入门者学习。因为简单，刚开始时获得众人的青睐，但好景不长，原因是它过于简单反而不切实用。

1991 年，芬兰赫尔辛基大学(University of Helsinki)计算机系的一名学生 Linus Torvalds 开始使用 Minix。他对 Minix 很感兴趣，但对 Minix 提供的功能不满意，因此打算改写 Minix 操作系统的最核心的部分——内核(Kernel)。他把所编写的内核的源代码放在 Internet 上，命名为“Linux”，允许人们自由使用，前提是遵守 GNU GPL 许可证。

Torvalds 也邀请其他程序员加入开发行列。很快地，许多人开始修改及加强 Linux。如今，Linux 除了可以在原先设计的 Intel x86(x>=3) 平台上执行外，也被移植到 Alpha、Sun Sparc、Motorola 68K、MIPS 以及 PowerPC 等平台上。

Torvalds 开发和维护的是 Linux 内核。内核是所有 Linux 操作系统的核心，是最重要的部分。自 1991 年发布版本 0.02 的内核到 1994 年发布版本 1.0 的内核，Linus Torvalds 的开发工作都是稳步进行。随着 Internet 的普及发展，越来越多的开发人员加入到 Linux 的开发行列，使得 Linux 不断改善提高，日益完善。

目前 Linux 内核最新的稳定版本是 2.6.11。

【注】 Linux 内核的版本号的第二位（例如上述的“6”）为偶数表明这是一个可以使用的稳定版本；版本号的第二位为奇数的版本一般有一些新的特性加入，是一个不一定很稳定的测试版本，一般供内核的开发人员或一些爱好者作测试使用。

1.1.5 Linux 发行版本

用户要利用 Linux 来使用计算机的功能，并不只是依赖一个软件程序“内核”就足够。除了内核，通常还需要数量众多的常用软件。这必须由某个人或某个机构来收集，并将一些常用的软件与 Linux 内核“包装”在一起以方便用户安装和使用。通常称之为“Linux 发行版本”。如果所包含的软件都是像 Linux 内核一样的自由软件，则该发行版本一般称之为“GNU/Linux”，例如“Debian GNU/Linux”。

每个用户都可以发布属于自己的发行版本，因为里面包含的软件通常都是自由软件，任何人都可以增加或减少其中的某些软件，从而形成一个特别的版本。但要发布一个真正有自己鲜明特色的版本并不那么简单，首先需要花费时间和精力去搜集组织需要用到的自由软件，然后一一测试通过，以后还要时刻关注各个软件的新发展并更新原来的旧版本（有时候在某个软件中的一个小小的改进也会引起与其他软件之间的依赖性或冲突等兼容性问题）。因此，大部分用户都是直接选用某些组织或公司所发布的 Linux 发行版本。

常见的一些 Linux 发行版本有：

- Debian Linux [http://www.debian.org]
- Slackware Linux [http://www.slackware.org]

- **Mandrake Linux** [http://www.mandrakelinux.com]
- **Redhat Linux** [http://www.redhat.com]
- **Suse Linux** [http://www.suse.com]
- **Turbo Linux** [http://www.turbolinux.com]

1.2 安装 Linux 系统

【问题的提出】

Geecy 软件开发公司在真正转移到使用 Linux 作为开发平台之前，需要安装一个适合使用的 Linux 系统。但 Linux 的版本繁多，应该如何选择才算是“适合使用”？选定某个版本后，又应该如何安装？

【问题分析】

虽然各个 Linux 发行版本都是基于 Linux 内核，但是每个发行版都有各自偏重的方面。如何选择发行版本必须结合使用者的因素（诸如使用 Linux 系统的目的、使用者的专业水平等）以及发行版本的特点（诸如系统风格、系统稳定性等）进行考虑，权衡选择。

但是，不管选择哪个 Linux 发行版本，安装及使用 Linux 系统原则上是大同小异的。也不必担心选错版本而在各种版本之间徘徊不定，因为用户在一个 Linux 发行版本的使用经验绝大部分都可以重用或迁移到另一个 Linux 发行版本上。

使用 Linux 系统的用户可以分为两大类：

- 系统管理员(Administrator)，负责整个 Linux 系统的安装和日常维护（诸如升级软件、管理用户、配置网络等）；
- 普通用户(User)，只使用 Linux 系统进行一般性的操作（诸如个人文档的管理、应用软件的开发等）。

对于系统的部署，有两种基本的方案：

- 集中式安装和管理，即需要配备一名专门的系统管理员在一台计算机上安装 Linux 系统并负责以后的日常维护工作，其他普通用户只要在自己的开发平台上（诸如 Microsoft Windows 等熟悉的平台）通过网络登录到该 Linux 系统来进行开发工作；
- 独立安装和管理,即为每个需要使用 Linux 系统的开发人员都安装一个独立的 Linux 系统，他们每个人既是系统管理员，又是普通用户。

不管是哪一个方案，都可以有两种选择：

- 使用物理上独立的计算机（即通常所说的计算机）进行安装；
- 使用称为虚拟计算机(Virtual Machine)的软件在某种平台上（诸如 Microsoft Windows 等熟悉的平台）模拟一台计算机裸机进行安装。

1.2.1 确定系统需求

Geecy 软件开发公司拥有近百名的软件开发人员。将要建立的 Linux 系统主要为开发人员的日常开发工作服务，因此必须能够满足各种开发过程的要求：

- 总体拥有成本低，管理维护简单；

- 能够方便地在开发人员之间共享使用公司的开发文档、源代码以及参考资料等资源；
- 提供功能强大且简单易用的适合开发人员使用的文本编辑器；
- 提供必要的开发环境（诸如 C、C++以及 Perl 等）；
- 要求系统具有足够的稳定性；

1.2.2 选择发行版本

目前，Linux 的发行版本种类繁多，就算是被认为“主流”的发行版本也有不少（请参考“1.1.5 Linux 发行版本”）。

一般来说，各个 Linux 发行版本之间是“源代码级”兼容的，即同一个源程序（诸如 C 或 C++）可以在各个发行版本正确编译和运行。但各个 Linux 发行版本仍具有一定的差异，主要表现在：

- 采用的内核

有些发行版本偏向于追求最新版本号的内核，而另一些发行版本则不求最新只求稳定。

- 版本的性质

有些发行版本完全由某些自由团体发行，比如著名的 Debian Linux，也称之为 Debian GNU/Linux，因为该发行版本包含的软件大部分都是自由软件。也就是说，Debian GNU/Linux 的发行方式与 Linux 内核的发行方式一致，其行为不会受制于某人或某商业公司。用户所需要的技术帮助都可以从 Debian 的网站获得。

另一些发行版本则由某些商业公司控制发行，例如 Redhat Linux。为方便用户安装和维护系统，商业公司一般会加入一些自主开发的工具软件，并提供一些收费的技术支持服务以满足某些用户的需求。

- 系统工具

系统安装程序、软件升级和卸载工具以及系统配置工具等都可能因发行版本的不同而有差异。例如，在系统配置工具方面，Turbo Linux 中这类工具一般以“turbo”开头命名，而在 Redhat Linux 中则一般以“redhat”开头命名。

- 系统目录结构

系统配置文件一般是分门别类地存放在目录“/etc”下以及该目录的各级子目录下，但具体的存放位置有时候又因为发行版本的不同而有一些小小的差异。

综合各种因素考虑，由于 Debian GNU/Linux 具有众多的优点：

- 从 1993 年开始发行，历史悠久，得到广大用户的认可；
- 其目标是致力于打造一个稳定的系统，不轻易添加新技术；
- 大部分软件都来源于 GNU 计划，是一个完全自由的操作系统；
- 安装过程简洁，基本系统精小，可以由用户根据自身需要自由定制和扩展；
- 系统稳定，性能可靠，应用软件丰富，使用简单方便；
- 由 Debian 网站提供足够的技术文档、使用帮助以及其他形式的帮助；

而且能够满足现有的需求，因此 Geecy 软件开发公司决定选用 Debian GNU/Linux。

Debian GNU/Linux 当前的稳定(Stable)版本是 3.0 版。Debian GNU/Linux 3.0 (代号 woody)

发行于 2002 年 7 月 19 日，至今已经作了多次的修正。最近一次（第 4 次）修正的发行版本是 Debian GNU/Linux 3.0r4，修正于 2005 年 1 月 1 日。其次是 Debian GNU/Linux 3.0r3，修正发行于 2004 年 10 月 28 日。

1.2.3 确定部署方案

由于 Geecy 软件开发公司拥有近百名的软件开发人员，如果要求每个开发人员独立安装和管理一个 Linux 系统，那么系统维护成本和员工培训成本将会非常高，而且也费时费力。

Linux 系统是一个真正的多用户多任务的操作系统。如果 Geecy 软件开发公司采用集中式安装和管理一个 Linux 系统，那么除了能够充分享受 Linux 系统的多用户特性的好处外，还能够显著降低成本和简化系统管理维护工作，同时也能够便于员工合作，提高工作效率。

Linux 系统是整个公司的业务关键所在，因此应该使用一台独立的计算机专门用于 Linux 系统的运行，并指派至少一位员工担任系统管理员，以保证系统的正常运作。

Geecy 软件开发公司最后决定采用集中式安装和管理一个 Linux 系统的方案，并指定一名曾经做开发工作的员工 Ray 担任 Linux 系统管理员。

【提示】 如果开发人员希望在工作之余可以学习如何管理和维护一个 Linux 系统，或者是回到家里也能够继续使用 Linux 进行开发工作，那么这些开发人员可以在原有的 Microsoft Windows 系统上安装一个虚拟机软件（例如 Vmware Workstation，http://www.vmware.com）从而模拟出一台计算机裸机用于安装 Linux 系统。

1.2.4 进入安装过程

下面介绍的安装过程对于使用物理上独立的计算机和使用虚拟机进行安装都是适用的。假设计算机的主要硬件参数如下：

中央处理器(CPU)　：Intel x86 兼容 CPU（x 不小于 4）
内存(Memory)　：容量 128 MB
硬盘(Hard Disk)　：IDE 接口、容量 10GB
网卡(Network Card)　：10/100Mbps 自适应 PCI 网卡
光驱(CD-ROM)　：IDE 接口光驱
显示器(Monitor)　：VGA 显示器
键盘(Keyboard)　：101 键兼容键盘

规划配置 Linux 系统使用的网络参数如下：
主机名(Hostname)　：geecy
IP 地址(IP Address)　：192.168.1.254
子网掩码(Netmask)　：255.255.255.0

整个安装过程比较简单，除了某些地方需要用户输入一些必要的信息（诸如 IP 地址、密码等）之外，大部分设置可以使用默认值，只需要用户按一个回车键（共计大约八十多次）确认即可进入下一步。

■ 准备安装光盘

Debian GNU/Linux 可以采用多种方式进行安装，例如从光盘、软盘、硬盘或者通过网络等方式都可以实现系统的安装。

在各种安装方式中，以光盘安装方式最为简单高效。用户可以通过访问 Debian 的主站点（http://www.debian.org）选择一个下载速度较快的镜像站点(Mirror Site)下载 Debian GNU/Linux 安装光盘的映像文件（ISO 文件），然后刻录成数据光盘。用户也可以选择直接购买整套产品光盘。

【提示】 发行版本 Debian GNU/Linux 3.0r3 一共有 7 张光盘，包含的软件包多达 8710 个。其中，第一张光盘包含有引导安装程序、基本系统以及大部分用户所需的最重要最常用的大量软件。如果是下载 ISO 文件，只需下载第一张光盘的映像文件（debian-30r3-i386-binary-1.iso）就可以满足大部分用户的应用需要。以后如果有需要再考虑下载剩余的光盘映像文件。如果是使用虚拟机进行安装，可以不用刻录。

■ 启动计算机

首先进入计算机的 BIOS（Basic Input Output System，基本输入输出系统）设置程序，修改系统的引导顺序，设置为 CD-ROM 优先引导启动。

不同版本的 BIOS 其设置程序的运行方式可能会有差异。多数版本的 BIOS 是在计算机开机自检时候通过按<Del>、<F1>和<F2>三个键中的其中一个就可以运行设置程序。不同版本的 BIOS 其设置程序界面会有一些差异。图 1-2 是 PhoenixBIOS 设置程序的界面，以供参考。

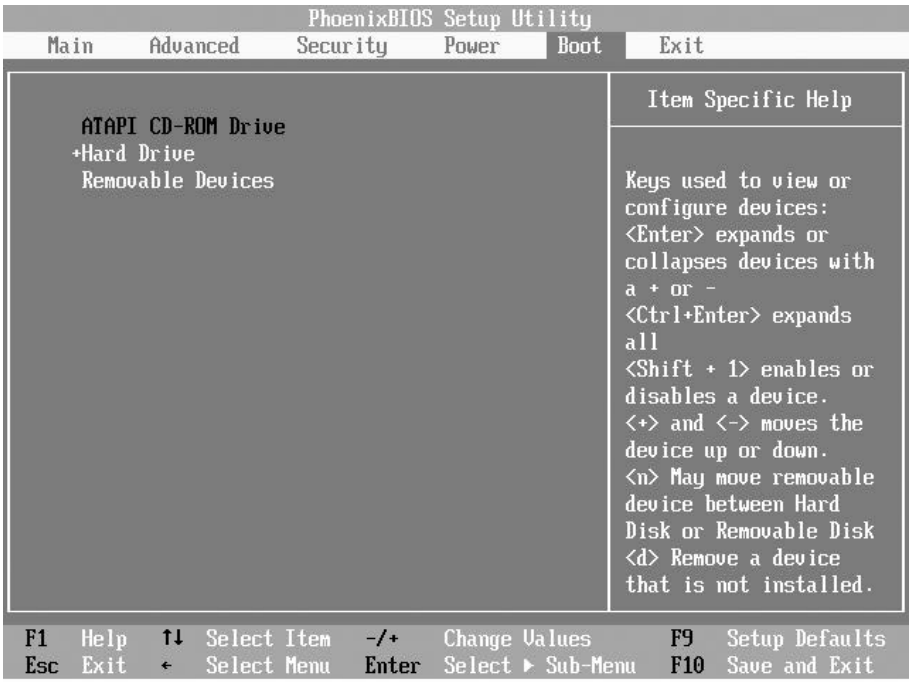

图 1-2

然后放置第一张安装光盘到光驱。最后保存对 BIOS 设置程序所作的修改，并退出。如无意外，计算机重新启动后应该可以从该安装光盘引导运行安装程序，并显示 Debian GNU/Linux

3.0 发行版本安装程序的初始界面（图 1-3）。

【提示】 若使用虚拟机安装，则既可以使用光盘，也可以把光盘的 ISO 文件关联到虚拟光驱。

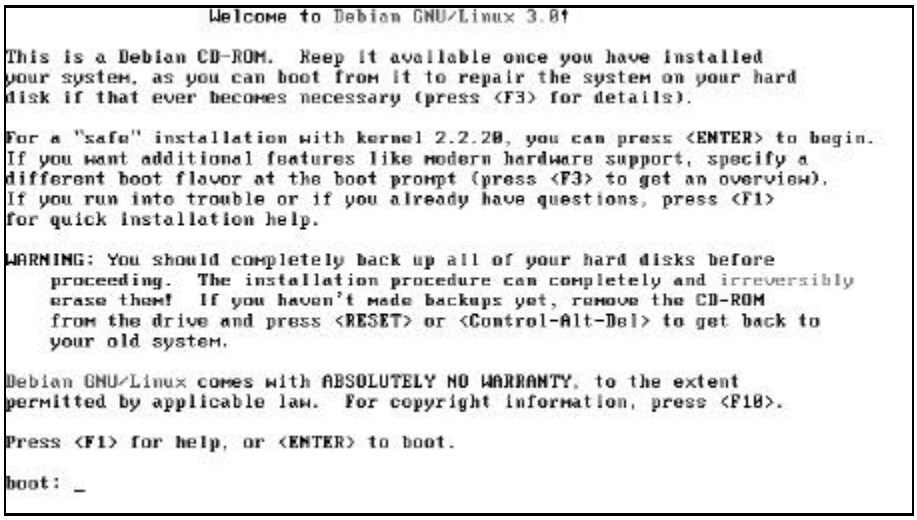

图 1-3

■ 安装开始

[boot]：首先是选择系统引导方式（图 1-3）。用户直接按回车键确认，安装程序选用默认的内核（版本号为 2.2.20）引导系统。

[Choose The Lanugage]：选择语言（图 1-4）。使用方向键“ ↓ ”向下移动光标选择“ en ”，然后按回车键确认。在接下来的安装阶段，安装程序将使用英文语言的安装界面。目前该稳定版本的安装程序暂时还没有简体中文的安装界面提供，只有测试版本提供测试。因此本示例选用英文界面。

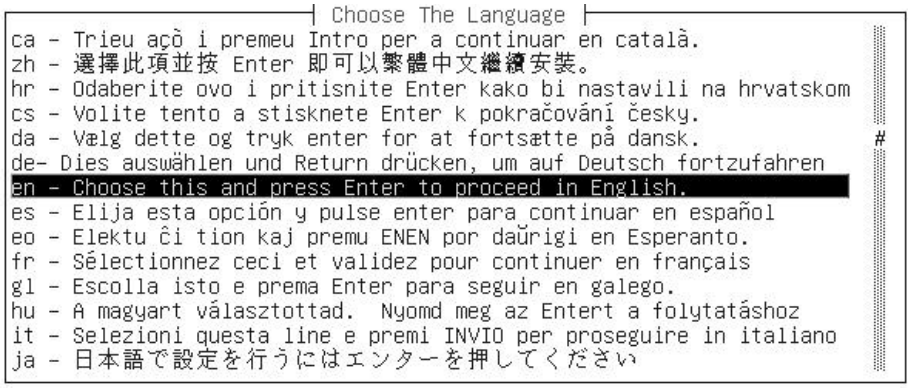

图 1-4

【提示】 本安装指导约定为：使用粗体的安装步骤表示需要用户使用键盘输入某一内容或修改某一选项，例如“ [choose The Language] ”；没有使用粗体的安装步骤表示只需要用户直接按回车键确认（使用默认值）即可以进入下一个安装步骤，例如“ [boot] ”。

[Choose Language Variant]：选择语言类别。使用默认值(United States)。

[Release Notes]：发行说明。

[Debian GNU/Linux Installation Main Menu]：Debian GNU/Linux 安装主菜单。使用默认值(Configure the Keyboard)。

[Select a keyboard]：选择键盘。使用默认值(U.S.English)。

■ 硬盘分区

[Debian GNU/Linux Installation Main Menu]：Debian GNU/Linux 安装主菜单。使用默认值(Partion a Hard Disk，硬盘分区)。

[Select Disk Drive]：选择硬盘。使用默认值(/dev/hda)。

[LILO Limitations]：LILO 的限制信息。

然后安装程序对硬盘进行检测，检测结果分为两种。一是，该硬盘已经有分区表（可能曾经安装过某些操作系统)。在这种情况下安装程序将直接进入下一步（图 1-6)。

二是，该硬盘还没有分区表（硬盘被低级格式化、分区表被手工清除或者是新硬盘等都可以导致没有分区表)。那么安装程序将提示：

```
No partition table or unknown signature on partition table
Do you wish to start with a zero table [y/n] ?
```

用户直接按“ y ”键确认。然后安装程序进入下一步。（图 1-5）

[cfdisk 2.11n]：cfdisk 是 Debian GNU/Linux 提供的硬盘分区工具。图 1-5 是硬盘分区工具 cfdisk 读取一个容量为 10GB 的新硬盘（没有分区表）的分区信息并显示的界面。

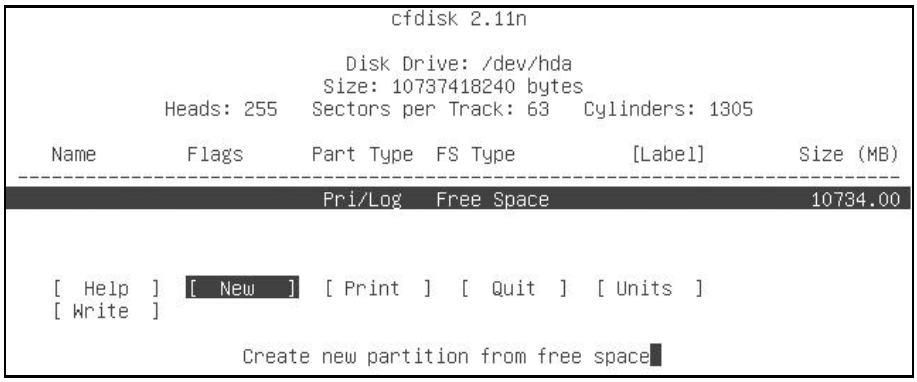

图 1-5

如果是硬盘具有分区表的情况，那么所显示分区的界面与图 1-5 将会有一些差异，主要是屏幕将会列出所有已经存在的分区的信息。如图 1-6 所示，是一个容量为 10GB 的硬盘曾经安装过某个 Linux 发行版本的分区例子。这时候用户可以通过先移动上下方向键“ ↑ ”和“ ↓ ”选中某个分区，再移动左右方向键 “ ← ” 和 “ → ” 选中[Delete]按钮，然后按回车键确定，以删除所选定的分区。重复此操作，删除剩余的所有分区，直到没有分区为止（如图 1-5 所示)。也就是说，接下来的安装步骤是基于图 1-5 所示的没有分区的情况进行的。

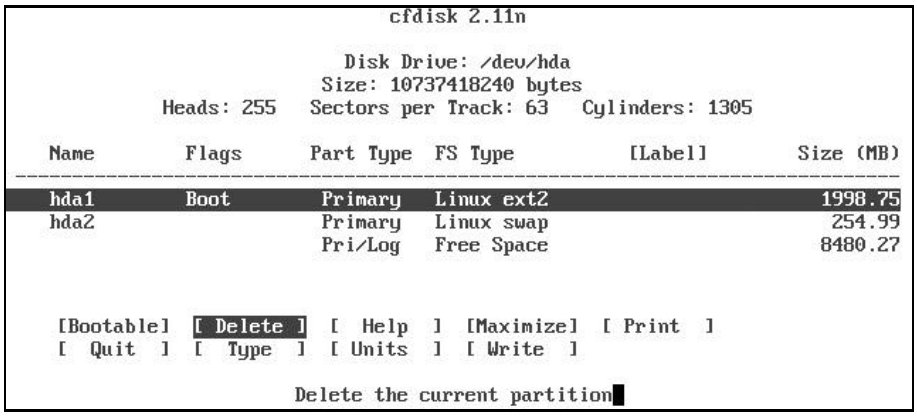

图 1-6

然后建立一个文件系统类型(FileSystem Type) 为“Linux”的分区（从图 1-5 开始）：

(1) 移动左右方向键“←”和“→”选中[New]按钮，按回车键确认；

(2) 使用默认值([Primary])，按回车键确认；

(3) 在 Size(in MB)提示信息后输入“2000”，按回车键确认；

(4) 使用默认值([Beginning])，按回车键确认；

(5) 使用默认值([Bootable])，按回车键确认；

经过这几步操作后，将成功建立一个可引导(Bootable)的、容量为 2000MB 以及文件系统类型为“Linux”的主分区(Primary Partition)。该分区在 Linux 系统的名字称为“hda1”。该步骤完成后，硬盘分区工具的界面变为如图 1-7 所示的界面。

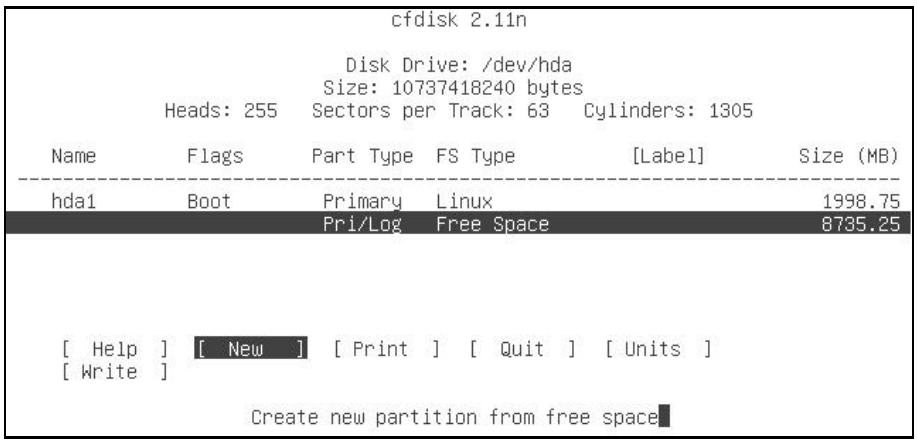

图 1-7

【提示】 文件系统类型为“Linux”的分区用于存放整个 Linux 系统的所有文件。如果完全按照本安装指导的操作步骤进行安装，那么需要的硬盘空间不超过 300MB。因此这里建立分区使用的容量 2000MB 只作参考。用户可以根据所使用的硬盘总容量选择一个值，建议是 500MB 至 3000MB。

接着还需要建立一个文件系统类型为“Linux swap”的分区（类似前面的步骤）：

（1） 移动上下方向键“ ↑ ”和“ ↓ ”选中剩余硬盘空间“ Free Space ”；

（2） 移动左右方向键“ ← ”和“ → ”选中[New]按钮，按回车键确认；

（3） 使用默认值([Primary])，按回车键确认；

（4） 在 Size(in MB)提示信息后输入“ 256 ”，按回车键确认；

（5） 使用默认值([Beginning])，按回车键确认；

经过这几步操作后，将成功建立一个容量为 256MB、文件系统类型为“ Linux ”的主分区。该分区在 Linux 系统的名字称为‘ hda2 ”。接下来还需要把分区 hda2 的文件系统类型由‘ Linux ”修改为“ Linux swap ”：

（1）移动左右方向键“ ← ”和“ → ”选中[Type]按钮，按回车键确认（确认之前，请确保所选中的分区是 hda2，如果不是，请移动上下方向键“ ↑ ”和“ ↓ ”选中分区 hda2）；

（2）将显示文件系统类型代码与文件系统类型的对照列表（请注意，82 就是需要修改的目标类型的代码），按回车键确认；

（3）将显示对照列表的剩余部分，使用默认值(82)，直接按回车键确认；

经过这几步操作后，分区 hda2 的文件系统类型已经由“ Linux ”修改为“ Linux swap ”，该步骤完成后，硬盘分区工具的界面变为如图 1-8 所示的界面。

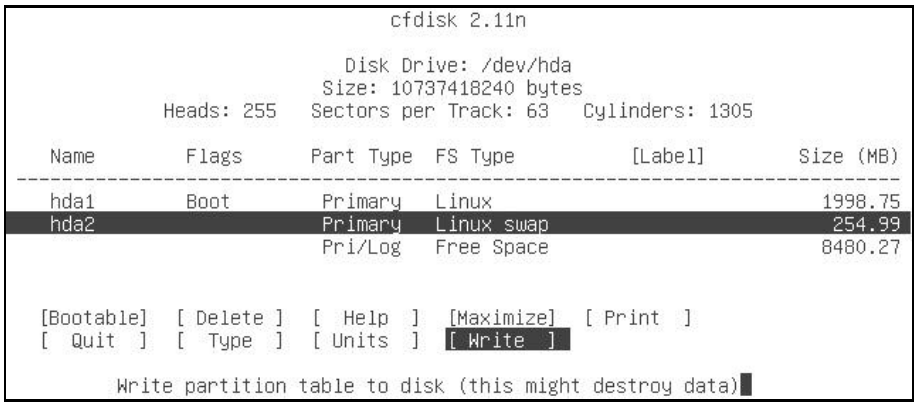

图 1-8

【提示】 文件系统类型为“ Linux swap ”的分区是 Linux 系统用于内存交换的分区。内存交换分区用于实现虚拟存贮管理，以解决物理内存容量的不足，类似于 Microsoft Windows 系统的内存交换文件。因此这里建立交换分区使用的容量 256MB 只作参考。用户可以根据物理内存容量和使用经验选择一个值。建议选用不小于物理内存容量的值。

最后，需要保存对硬盘所作的修改，并退出硬盘分区工具 cfdisk：

（1）移动左右方向键“ ← ”和“ → ”选中[Write]按钮并按回车键确认，将出现提示信息：

```
Are you sure you want write the partition table to disk? (yes or no): yes↙
```

用户输入 yes 并按回车键确认。保存成功后再次返回到图 1-8 所示的界面。

（2）移动左右方向键“ ← ”和“ → ”选中[Quit]按钮并按回车键确认，成功退出硬盘分区工具 cfdisk，返回到安装主菜单。

■ **初始化和激活交换分区**

[Debian GNU/Linux Installation Main Menu]：Debian GNU/Linux 安装主菜单。使用默认值(Initialize and Activate a Swap Partition，初始化和激活交换分区)。

[Scan for Bad blocks?]：需要检测坏扇区吗？使用默认值(No，不需要)。

[Are You Sure?]：你确定吗？使用默认值(Yes，确定)。

■ **初始化 Linux 分区**

[Debian GNU/Linux Installation Main Menu]：Debian GNU/Linux 安装主菜单。使用默认值(Initialize a Linux Partition，初始化 Linux 分区)。

[Scan for Bad blocks?]：需要检测坏扇区吗？使用默认值(No，不需要)。

[Are You Sure?]：你确定吗？使用默认值(Yes，确定)。

[Mount asthe Root Filesystem?]：要挂载成为根文件系统吗？使用默认值(Yes，确定)。

■ **安装内核和驱动程序模块**

[Debian GNU/Linux Installation Main Menu]：Debian GNU/Linux 安装主菜单。使用默认值(Install Kernel and Driver Modules，安装内核和驱动程序模块)。

[Found a Debian CD-ROM]：找到一张 Debian 安装光盘。使用默认值(Yes，确定)。

[Debian GNU/Linux Installation Main Menu]：Debian GNU/Linux 安装主菜单。使用默认值(Configure Device Driver Modules，配置设备驱动程序模块)。

[Note about loaded drivers]：关于已经装载的驱动程序的说明。

[Select Category]：选择分类。使用默认值(Finished. Return to previous menu.，完成并返回前一菜单)。

■ **配置网络**

[Debian GNU/Linux Installation Main Menu]：Debian GNU/Linux 安装主菜单。使用默认值(Configure the Network，配置网络)。

[Choose the Hostname]：选择主机名。输入“ geecy ”代替安装程序的默认值“ debian ”，然后按回车键确认。

[Automatic Network Configuration]：自动网络配置。移动左右方向键“ ← ”和“ → ”选中[No]按钮（不需要自动配置）并按回车键确认。

[Choose the IP Address]：选择 IP 地址。输入“ 192.168.1.254 ”代替安装程序的默认值“ 192.168.1.1 ”，然后按回车键确认。

[Choose the Network Mask]：选择网络掩码。使用默认值(255.255.255.0)。

[What is you IP gateway address?]：你的 IP 网关地址是什么？可以置空。使用退格键删除安装程序的默认值“ 192.168.1.1 ”，然后按回车键确认。

[Choose the Domain Name]：选择域名。直接按回车键确认。

[Choose the DNS Server Addresses]：选择 DNS 服务器地址。可以置空。使用退格键删除安装程序的默认值“ 192.168.1.1 ”，然后按回车键确认。

■ **安装基本系统**

[Debian GNU/Linux Installation Main Menu]：Debian GNU/Linux 安装主菜单。使用默认值(Install the Base System，安装基本系统)。

然后安装程序从安装光盘复制基本系统文件到硬盘。

■ 设置系统可引导

[Debian GNU/Linux Installation Main Menu]：Debian GNU/Linux 安装主菜单。使用默认值(Make System Bootable，设置系统可引导)。

[Debian GNU/Linux Installation Main Menu]：Debian GNU/Linux 安装主菜单。使用默认值(Install LILO in the MBR，安装 LILI 到主引导记录)。

[Securing LILO]：提示加强 LILO 的安全。

■ 重新启动系统

[Debian GNU/Linux Installation Main Menu]：Debian GNU/Linux 安装主菜单。移动上下方向键“ ↑ ”和“ ↓ ”选中(Reboot the System，重新启动系统)，然后按回车键确认。

[Reboot the System?]：重新启动系统吗？使用默认值(Yes，确认)。

【提示】 必须再次进入计算机的 BIOS 设置程序，修改系统的引导顺序，恢复设置为硬盘(Hard Disk)优先引导启动。否则，计算机会再次从光盘引导安装程序，重复上面的安装步骤，即屏幕会再次显示图 1-3 所示的界面。设置为硬盘引导启动后，保存对 BIOS 设置程序的修改并重新启动，然后由硬盘引导运行刚安装好的基本 Linux 系统，就可以继续完成剩下的系统配置的步骤。

■ 配置时区

[Debian System Configuration]：Debian 系统配置的提示信息。直接按回车键确认。

[Time Zone Configuration]：时区配置。移动左右方向键“ ← ”和“ → ”选中[No]并按回车键确认。接着移动上下方向键“ ↑ ”和“ ↓ ”选中“ Asia ”并按回车键确认。最后移动上下方向键“ ↑ ”和“ ↓ ”选中“ Shanghai ”并按回车键确认。

■ 配置密码(Password setup)

（1）设定系统的密码策略：

[Shall I enable md5 passwords?]：是否使用 MD5 算法对密码加密？使用默认值(No，不使用)。

[Shall I enable shadow passwords?]：是否使用影子密码？使用默认值(Yes，使用)。

（2）设定系统管理员的密码：

[Enter a password for root:]：设定管理员用户 root 的密码。输入密码，然后按回车键确认。

[Re-enter password to verify:]：再次输入密码用于校对。输入同样的密码，然后按回车键确认。

【提示】 系统管理员帐号的用户名默认是 root，其密码由用户在安装系统过程中设定。请牢记该密码，系统安装完成后如果要进行系统管理的任务，就需要使用管理员用户名 root 和该密码登录到系统。为安全起

见，用户输入的密码都不显示在屏幕上。密码区分大小写。关于如何选用密码，请参考“1.3.3 使用简单的命令”。

（3）创建普通用户帐号：

[Shall I create a normal user account now?]：现在需要创建一个普通用户帐号吗？使用默认值(Yes，使用)。

[Enter a username for your account:]：输入帐号的用户名。输入用户名“ray”，然后按回车键确认。

[Enter a full name for the new user:]：输入该用户的全名。使用默认值(Debian User)，直接按回车键确认。

[Enter a password for the new user:]：设定新用户(ray)的密码。输入密码，然后按回车键确认。

[Re-enter password to verify:]：再次输入密码用于校对。输入同样的密码，然后按回车键确认。

【提示】 系统管理员应该尽量避免使用 root 登录使用 Linux 系统，以防止某些误操作对系统产生破坏作用。应该在安装系统过程创建一个普通用户帐号，平时主要使用该帐号登录系统，必要时才切换到 root 用户登录。为安全起见，用户输入的密码都不显示在屏幕上。不但密码区分大小写，而且用户名也区分大小写。即系统认为 Ray 与 ray 是两个不同的用户。

■ 其它配置

[Shall I remove the pcmcia packages?]：需要移除 pcmcia 软件包吗？使用默认值(Yes，移除)。

[Do you want to Use a PPP connection to install the system?]：需要使用 PPP 网络连接来安装系统吗？使用默认值(No，不使用)。

[Scan another CD?]：搜索另外的安装光盘吗？使用默认值(No，不搜索)。

[Add another apt source?]：增加另外的 apt 资源吗？使用默认值(No，不增加)。

[Use security updates from security.debian.org?]：使用站点 security.debian.org 的安全更新吗？移动左右方向键“←”和“→”选中[No]并按回车键确认。

[Run tasksel?]：运行实用程序 tasksel 吗？使用默认值(Yes，运行)。

[Select tasks to install]：选择安装任务（图 1-9)。首先移动上下方向键“↑”和“↓”选中“[ ] C and C++”，再按空格键选择该项目（显示符号“*”表示被选择)，然后移动左右方向键“←”和“→”选中[Finish]并按回车键确认。

【提示】用户可以在基本系统的基础上选用更多的应用软件。这里只选用一个“C and C++”的开发工具，再配合基本系统中提供的实用软件，完全可以支持本书内容的学习。额外在选用其他的应用软件将会占用更多的硬盘空间。建议用户以后有需要才补充安装相应的应用软件。

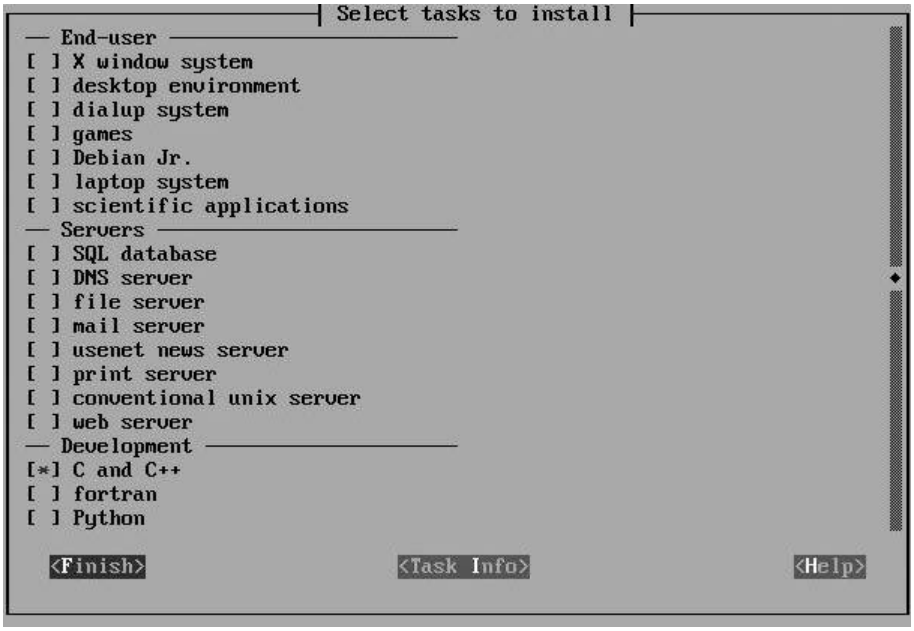

图 1-9

[Run dselect?]：运行实用程序 dselect 吗？使用默认值(No，不运行)。

接着，安装程序提示：

```
Do you want to continue? [Y/n] y↙
```

用户先按“y”键，然后按回车键确认。

接着，安装程序提示插入第一张安装光盘到光驱（要保证安装光盘还在光驱里面）：

```
Media Change: Please insert the disc labled 'Debian GNU/Linux 3.0r3 _Woody_ - Official
i38 Binary-1 (20041027)' in the driver '/cdrom/' and press enter
```

用户直接按回车键确认。

[Configuring Binutils]：配置 Binutils 的信息。

[Configuring Less]：配置 Less 吗？使用默认值(No，不配置)。

[Configuring Locales]：配置 Locales。按<Tab>键选中[Ok]，然后按回车键确认。

[Configuring Locales]：配置 Locales。使用默认值(Leave alone)。

[Configuring Nfs-common]：配置 Nfs-common 的信息。

配置 SSH:

[Allow SSH protocol 2 only]：只允许 SSH2 协议。使用默认值(Yes，确定)。

[Privilege separation]：关于权限分离的说明信息。

[Do you want /usr/lib/ssh-keysign to be installed SUID root?]：使用默认值(Yes，确定)。

[Do you want to run the sshd server?]：运行 sshd 服务器吗？使用默认值(Yes，确定)。

配置 CVS:

[Where are your repositories?]：使用退格键删除安装程序的默认值“ /var/lib/cvs ”，然后按回车键确认。

[Should the CVS pserver be enabled?]：启用 CVS 吗？使用默认值(No，不启用)。

接着，安装程序解压缩和安装在图 1-9 中所选择的软件包。然后继续配置。

出现提示信息：

```
Select the number of the default dictionary [1]
```

直接按回车键确认。然后出现提示信息：

```
Do you want to erase any previously downloaded .deb files? [Y/n] y↙
```

用户先按“ y ”键，然后按回车键确认。

```
Press enter to continue.
```

按照此提示，直接按回车键确认。

```
[---Press return---]
```

按照此提示，直接按回车键确认。

```
Select a number from 1 to 5 from the list above.
Enter value (default='1', 'x' to restart): 5↙
```

输入“ 5 ”(不配置)，然后按回车键确认。

■ 安装完成

[Thank you for choosing Debian!]：感谢使用 Debian！

按回车键确认后，显示如下的登录界面：

```
Debian GNU/Linux 3.0 geecy tty1


geecy login:
```

表明整个安装过程完成，Debian Linux 已经成功安装。

1.3 使用 Linux 系统

【问题的提出】

Geecy 软件开发公司的系统管理员 Ray 已经成功安装了一个 Linux 系统。接下来应该从哪里开始？

【问题分析】

要充分使用 Linux 系统提供的功能，当然是对 Linux 越精通越好。但是 Linux 所涉及的领域和内容实在是太多，因此不能一蹴而就。

对于普通用户而言，当务之急是先成功登录到 Linux 系统，然后学习使用一些比较简单的命令，最后使用完毕需要安全的退出 Linux 系统。

对于系统管理员，还需要在这个基础上学习如何正常顺利的启动 Linux 系统和正确安全的关闭 Linxu 系统。

因此，Ray 应该先启动 Linux 系统，然后以普通帐户 ray 登录，并初步学习使用一些简单的命令，接着退出此次的登录，最后学习如何安全关闭 Linux 系统。

1.3.1 启动 Linux

Debian GNU/Linux 3.0 使用的启动管理器（Boot Loader）称为 LILO（LInux LOader)。启动管理器能够引导多个（多种）操作系统启动，一般是在启动前提供一个操作系统的菜单列表供用户选择其中一个进行引导启动。

Geecy 软件开发公司是在单独的计算机上安装 Linux 系统，因此无需选择。屏幕开始显示信息“LILO 22.2 Loading Linux....”表明计算机正在加载 Linux 内核。

Linux 内核加载到系统内存中，内核对计算机的硬件进行检测，然后加载 Linux 预先设定自动加载的模块，最后启动 Linux 预先设定自动运行的服务。

每成功启动一项服务，屏幕将打印一条带有“[OK]”的信息，如果失败，将打印带有“[Failed]”的信息。作为管理员，应该特别留意那些失败的信息。

大多数时候，这些服务都会正常启动。就算有一些是失败信息，只要不是致命错误，也可以过关。在系统正常启动完成后，应该显示如下的登录界面：

```
Debian GNU/Linux 3.0 geecy tty1

geecy login:
```

表明 Linux 系统已经启动成功，用户可以登录使用 Linux 系统。

1.3.2 登录 Linux

系统管理员 Ray 拥有两个帐号。先使用普通帐号 ray 登录，输入用户名 ray：

```
geecy login: ray↙
Password:
```

输入“ ray ”后按回车键，然后系统要求输入密码。用户 ray 的密码是在安装 Linux 系统的过程中由安装人员设定的。（不会忘了吧？）

接着输入密码。需要注意，为了安全起见，密码不显示在屏幕上，而且也没有像“ * ”之类的符号。如果用户名与密码都被 Linux 系统验证通过，那么将出现 Linux 系统默认的命令提示符：

```
ray@geecy:~$
```

表明用户 ray 已经登录成功，可以开始执行 Linux 的命令。

Linux 系统默认的命令提示符包含了一些重要的信息：

- “ ray@geecy ”事实上就是现在所流行的电子邮件地址的格式，即 ray 是用户名（在此则表示登录时使用的用户名），geecy 是网络地址（在此则表示 Linux 系统的主机名）；
- 最后的一个“ ~ ”表示用户当前的工作目录（请参考“ 3.1　目录基本操作 ”）；
- 符号“ $ ”表明当前的用户是普通用户，没有权限进行系统管理工作。因此 Ray 可以放心学习一些简单的命令，不用担心因为误使用了某些命令而引起整个系统的崩溃。

1.3.3 使用简单的命令

- 关于密码

这是 Ray 第一次使用 Linux 系统。安装系统的时候，Ray 为帐号设置了一个比较简单的密码，考虑到安全问题，Ray 需要修改普通用户 ray 的密码(Password)。修改密码使用 passwd 命令：

```
ray@geecy:~$ passwd↙
Changing password for ray
(current) UNIX password:
```

普通用户执行 passwd 命令执行后，系统要求用户先输入原来所是用的密码。Ray 输入了原来使用的密码。同样，为了安全起见，密码不显示在屏幕上。然后提示输入一个新密码：

```
Enter new UNIX password:
```

[提示] 应该如何为自己选择一个好的密码呢？好的密码应该自己容易记忆，别人猜测和暴力穷举破解困难。但这本身是一个矛盾的结合体。根据经验，现推荐一种简单而有效的选择密码的方法：首先，设想一个普普通通的句子，例如是“ 今天，天气晴朗。”，然后依次提取每个字词的首字母，连同句子中的标点符号一起构成一个别人猜测和暴力穷举破解将非常困难而自己容易记忆的密码：“ Jt,Tqql. ”。这个密码有 8 个字符长度。由于兼容性原因，请把中文状态的标点符号转换成英文状态的标点符号。为增加暴力穷举破解的难度，建议第一个字符大写，且总长度不小于 8。自此，用户只需要记忆一句话“ 今天，天气晴朗。”即可。

接着，Ray 输入了一个自己喜欢的新密码，输入完毕，回车。接着提示再次输入新密码：

```
Retype new UNIX password:
```

由于密码不显示在屏幕上，因此，为避免输入错误，系统要求用户再次输入新密码，系统将根据用户两次输入的密码是否一致，以确认用户的输入是否有误。

Ray 再次输入同样的密码，回车。系统提示密码更新成功：

```
passwd: password updated successfully
```

需要注意，如果用户在中途不希望继续修改密码，那么可以随时按组合键<Ctrl> + d 来提前结束 passwd 命令的执行。

[注意] 对于用户来说，修改密码是一件比较危险的事情。试想，如果 Ray 两次都打错了同样的一个或几个字符，那么将会发生什么？Ray 以后将不能再使用帐号 ray 登录到该 Linux 系统。虽然可以使用管理员帐号 root 登录系统为帐号 ray 设置一个新密码，但如果不是 ray 用户，而是 root 用户呢？为避免麻烦，Ray 应该在退出 Linux 系统前，验证新密码的设置是否如他所愿。方法是，在退出 Linux 系统前使用三个键的组合键<Ctrl> + <Alt> + <F2>，切换到第二个登录界面，使用用户名 ray 和新密码登录，如果登录失败，可以再使用三个键的组合键<Ctrl> + <Alt> + <F1>把屏幕切换到刚才修改密码的界面，再次执行 passwd 命令，重新设定一个新密码。待验证成功以后才退出 Linux 系统。

■ 关于日期和时间

使用 date 命令可以显示系统的当前日期和时间：

```
ray@geecy:~$ date↙
Fri Jan 28 07:20:39 CST 2005
```

本例中显示的时间是北京时间 2005 年 1 月 28 日星期五早上 7 时 20 分 39 秒。date 命令的默认显示格式是“ 星期 月 日 时:分:秒 时区 年 ”。

Linux 系统有一个功能强大的万年历 cal。这是一个真正意义的“ 一万年 ”的日历(Calendar)，可查的年份从公元 1-9999。

单独一个 cal 命令显示 Linux 系统时间的当天所在的月份的月历：

```
ray@geecy:~$ cal↙
    January 2005
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

查看任意年份的年历需要指定一个具体的年份（1-9999）作参数，例如：

```
ray@geecy:~$ cal 2010↙
```

将查看公元 2010 年全部 12 个月的年历。用户也可以同时指定年和月，例如：

```
ray@geecy:~$ cal 7 2010↙
```

将查看公元 2010 年 7 月份的月历。下面的查询可以让用户重新感受历史上的十多天“ 空

白”：

```
ray@geecy:~$ cal 9 1752↙
   September 1752
Su Mo Tu We Th Fr Sa
       1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

■ 关于计算器

Linux 系统下有一个功能十分强大的具有任意位精度的计算器 bc。直接输入命令 bc 启动该计算器：

```
ray@geecy:~$ bc↙
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.

```

首先输出一些版本号和版权信息，然后光标停留在最后一行的开始位置，等待用户输入待计算的表达式。例如输入“ 123+256 ”，然后回车，计算器将求出结果 379：

```
123+256↙
379
```

用户可以尝试输入一些比较复杂的表达式进行计算：

```
123456789*987654321↙
121932631112635269

( (1+2) * (20-3) - 4*5 ) * 123↙
3813

22/7↙
3

12345678901234567890123456789012345678901234567890123456789*123456789012345678
901234567890123456789012345678901234567890123456789012345↙
152415787532388367504953515625666819450083828733760097552251028798943606233105608908631685719395281054792739353764595060205
```

从上面例子看出，bc 对任意位整数的乘法可以精确计算出来，但对 22/7 的计算结果却没有保留小数位。然而用户可以使用 scale 变量设定计算结果所保留的小数位数目。例如：

```
scale=100
```

```
22/7↙
3.142857142857142857
```

```
Debian GNU/Linux 3.0 geecy tty1

geecy login: root↙
```

root 登录成功后就可以执行：

```
geecy:~# shutdown -h now↙
```

将立即准备关闭计算机。在关闭之前，shutdown 命令将把计算机内存的数据更新保存到硬盘，以及做一些其他必要的工作。等待屏幕提示关闭完毕管理员就可以切断电源。

shutdown 命令的选项(Option)“ -h ”代表停机(Halt)，另一个可选的选项是“ -r ”代表重启(Reboot)。参数“ now ”表示立即关机，不等待。可选的参数除了“ now ”还可以是具体的等待时间，以分钟为单位。例如：

```
geecy:~# shutdown -r now↙
```

将表示立即准备关闭计算机，并在关闭计算机后重新启动。即所谓的“ 热启动 ”。而

```
geecy:~# shutdown -h 10↙
```

将表示等待 10 分钟后自动关闭计算机。

至此，Ray 的第一次 Linux 之旅圆满结束。

1.4 配置 Linux 系统

【问题的提出】

Geecy 软件开发公司所安装 Linux 系统目前基本上是使用默认配置，不是很适合开发人员的使用。虽然 Linux 系统是真正的多用户系统，但在目前的默认配置下任何时刻只能有一个开发人员可以坐在安装有 Linux 系统的主机（以及键盘和显示器）前使用系统。

现在该公司要求系统管理员 Ray 对 Linux 进行一些必要的配置，使得所有的开发人员都能够在各自的计算机上（一般是安装 Microsoft Windows 系统）通过网络远程登录使用 Linux 系统。

【问题分析】

计算机的控制台(Console)是指直接连接到主机的输入设备(一般是键盘)和输出设备(一般是显示器)。

不管是让所有开发人员通过网络同时登录使用 Linux 系统还是让所有开发人员坐在 Linux 的控制台前一个接着一个的轮流登录使用 Linux 系统，Ray 都必须首先为每个开发人员建立和分配一个不重复的帐号（包含登录所用的用户名和密码)。有了帐号，开发人员就可以在 Linux 的控制台轮流使用 Linux 系统。

对这些普通帐号的用户名的选择，建议使用英文名作为用户名，或者使用姓名的拼音（姓的完整拼音加上名的第一个字母)。例如“ ray ”、“ tom ”、“ huangdf ”等。

Linux 系统和 Microsoft Windows 系统都可以支持使用 TCP/IP 协议进行网络通信。Ray 需要配置设定 Linux 系统在网络中所使用的 IP 地址，并对每一位开发人员所使用的 Microsoft Windows 系统分配一个符合要求的在同一个逻辑子网不相重复的 IP 地址。这样配置网络之后，所有的开发人员就能够在各自的计算机上通过网络远程登录使用 Linux 系统。

Internet 保留给局域网使用的 IP 地址有三个段：10.0.0.0/8 、 172.16.0.0/16 以及 192.168.0.0/16 。 Geecy 软件开发公司的规模并不是非常庞大，因此选用其中的 192.168.1.0/24 即是从 192.168.1.1 - 192.168.1.254 的共 254 个 IP 地址就可以为 Geecy 软件开发公司最多可以分配给 254 台计算机使用。

安装 Linux 系统的时候，已经把 IP 地址 192.168.1.254 分配给 Linux 系统。因此，其他开发人员的系统从 192.168.1.1 开始依次分配。

通过网络远程登录使用 Linux，历史上曾经流行使用 Telnet 软件，但由于 Telnet 协议使用明文传输信息，密码将暴露在网络途中，产生安全问题。当前的趋势是使用 Telnet 协议的改进版本——加密的 Telnet——SSH(Security Shell)协议。Debian GNU/Linux 3.0 默认设置开机自动运行的 SSH 服务软件是 OpenSSH，能够支持 SSH2 协议，默认监听的端口号是 22。

开发人员使用的 Microsoft Windows 系统，默认没有安装支持 SSH 协议的客户端软件。用户需要自行下载安装。比较流行好用的一个是 PuTTY，其二进制可执行程序和源代码都使用 MIT 许可证发布，能够在多种平台上运行，是开源软件，免费使用。用户可以到站点：

http://www.chiark.greenend.org.uk/~sgtatham/putty/

自行下载。目前其稳定版本的版本号是 0.55。PuTTY 在 Intel x86 平台的 Windows 版本的二进制可执行程序只有一个文件“putty.exe”，下载后无需安装，双击运行即可。

1.4.1 配置用户

创建用户帐号需要有管理员权限。系统管理员 Ray 虽然可以直接使用 root 用户登录，但是之前曾经建议，平时应该使用普通帐号 ray 登录。为了避免频繁的登录和退出操作，Linux 系统提供一个命令 su 可以达到简化操作的目的。

Ray 首先使用普通帐号 ray 登录成功：

```
ray@geecy:~$
```

然后使用 su 命令切换到 root 用户：

```
ray@geecy:~$ su -↙
Password: ↙
geecy:~#
```

需要输入管理员帐号 root 的密码。如果密码验证通过，那么立即出现 root 的提示符。选项“-”表示模拟真正的登录，即与直接使用 root 用户登录有相同的效果。然后就可以执行管理任务。

使用 useradd 命令增加用户帐号。例如：

```
geecy:~#  useradd -m tom↙
```

选项“-m”表示在创建帐号的同时自动为该用户建立相应的个人目录（/home/tom）。所增加的帐号还需要设定密码才能登录使用：

```
geecy:~#  passwd tom↙
```

passwd 命令使用一个用户名作为参数，将修改该用户的密码。设定密码需要两次输入一致，且输入的密码不回显。

需要注意，如果漏了用户名作参数，那么 passwd 命令要修改的将是当前登录的用户(即root)密码。

现在，Ray 已经为自己成功为开发人员 Tom 分配了一个普通帐号 tom。以同样的方式，Ray 为其他开发人员增加帐号，例如“bill”和“huangdf”等。

删除帐号可以使用 userdel 命令，例如：

```
geecy:~# userdel huangdf↙
```

将删除帐号 huangdf。用户以后将不能使用用户名 huangdf 登录 Linux 系统。如果需要把待删除用户的个人目录等私人资料连同帐号一起删除，可以使用“-r”选项：

```
geecy:~# userdel -r huangdf↙
```

如果管理任务完成，应该立即退出管理员帐号：

```
geecy:~# exit↙
ray@geecy:~$
```

1.4.2 通过控制台使用 Linux

Ray 已经成功为每一个使用者分配了一个独立的帐号。现在，每个用户可以轮流地在控制台前使用 Linux 系统。以下的经验对控制台的使用有很大帮助。

■ 回滚控制台屏幕

相信很多 Linux 系统管理员都有类似的经历，在启动 Linux 的过程中屏幕所打印的信息往往是一闪而过，昙花一现，根本来不及看清楚。但不必担心，Linux 系统提供了一个命令 dmesg(Display Message)可以让用户重温这些信息。例如，Ray 以普通帐号 ray 登录，然后执行该命令：

```
ray@geecy:~$ dmesg↙
```

启动信息虽然是重返屏幕了，但依然是一闪而过，而且速度更快！该怎么办？

“软”的方法用户可以参考“6.2 管道”。这里介绍一种“硬”的方法——利用控制台屏幕的“回滚”特性实现。

控制台屏幕的行数毕竟有限，因此 Linux 系统提供了简捷的方法控制屏幕的回滚，利用<Shift> + <PageUp>和<Shift> + <PageDown>这两个组合键来分别实现屏幕内容的向上和向下的滚动，从而实现能够看到那些被滚动上去的内容。使用方法是先按住<Shift>键不放，然后根据需要分别按<PageUp>键和<PageDown>键，即可以控制屏幕自由滚动。

■ 切换虚拟控制台

虽然一般只有一个硬件控制台（一套键盘和显示器）连接到 Linux 系统的主机，但 Linux 系统可以模拟出若干个控制台供用户随时切换使用。这些控制台称为虚拟控制台(Virtual Console)。Linux 系统默认配置有 7 个虚拟控制台（tty1-tty7），相当于为用户添置了 6 套键盘和显示器。一般情况下，用户所使用的控制台默认是 tty 1。

用户可以使用组合键<Alt> + <Fx>在各个虚拟控制台之间进行切换。其中，x 是由 1 到 7 的整数。例如，要切换到 tty3，用户可以使用组合键<Alt> + <F3>。

tty1 到 tty6 是文本控制台；tty7 是图形控制台，专用于运行 X Windows 系统（一个图形化的用户接口，本书不涉及图形用户界面方面的内容）。

Linux 系统拥有多个虚拟控制台，极大的方便了用户的工作。例如在“1.3.3 使用简单的命令”一节 Ray 已经提前享受了多个虚拟控制台带来的便利。

*1.4.3 配置网络

由于安装 Linux 系统的过程中已经为 Linux 系统分配了 IP 地址 192.168.1.254 以及子网掩码 255.255.255.0。因此只需要检查 Linux 系统的网卡是否真正使用该网络参数工作。

使用命令 ifconfig 查看网络配置：

```
ray@geecy:~$ ifconfig↙
eth0      Link encap:Ethernet  HWaddr 00:50:56:D3:8D:9E
          inet addr:192.168.1.254  Bcast:192.168.1.255  Mask:255.255.255.0
```

```
（省略后面显示内容）
```

在 Linux 系统中，eth0 是第一张网卡的名称。

然后使用下面的命令查看 SSH 服务是否正常运行：

```
ray@geecy:~$ netstat | grep ssh ↙
tcp        0      0 *:ssh                   *:*                     LISTEN
```

这表示 SSH 服务运行正常。关于该命令的使用，请参考“第 6 章　Linux 组合命令”。

然后，Ray 需要为开发人员所使用的每台计算机分配适当的 IP 地址。例如，开发人员 Tom 所分配到的 IP 地址是 192.168.1.2，那么他需要在其 Microsoft Windows 系统上修改网络配置（图 1-10）：

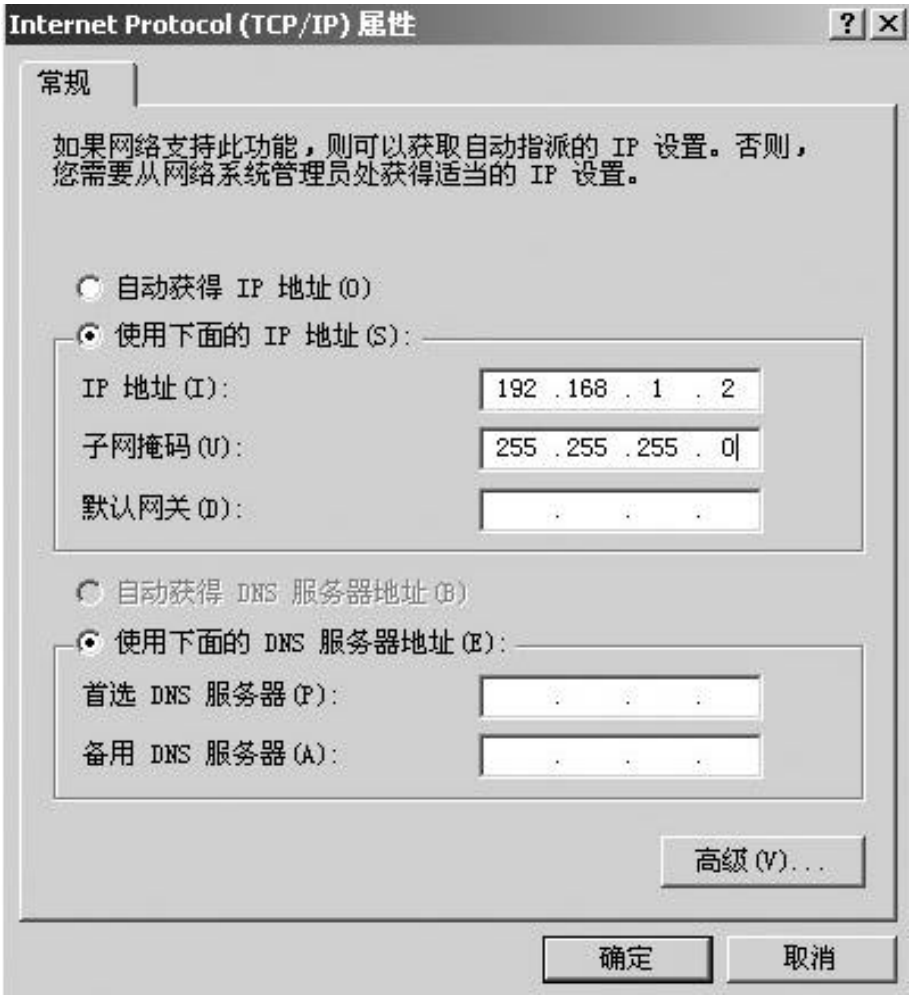

图 1-10

最后，Tom 在该 Windows 系统下使用 ping 命令测试网络连通性：

```
C:\> ping 192.168.1.254 ↙
```

如果命令运行结果类似于：

```
Reply from 192.168.1.254: bytes=32 time=1ms TTL=255
Reply from 192.168.1.254: bytes=32 time<1ms TTL=255
```

```
Reply from 192.168.1.254: bytes=32 time<1ms TTL=255
Reply from 192.168.1.254: bytes=32 time<1ms TTL=255
```

那么就表明 Tom 的 Windows 系统与 Linux 系统的能够正常连通。

*1.4.4 通过网络使用 Linux

Tom 的计算机的网络配置已经完成。现在，Tom 在 Windows 系统上使用 SSH 客户端软件 PuTTY 远程连接使用 Linux 系统。

PuTTY 的运行界面如图 1-11 所示：

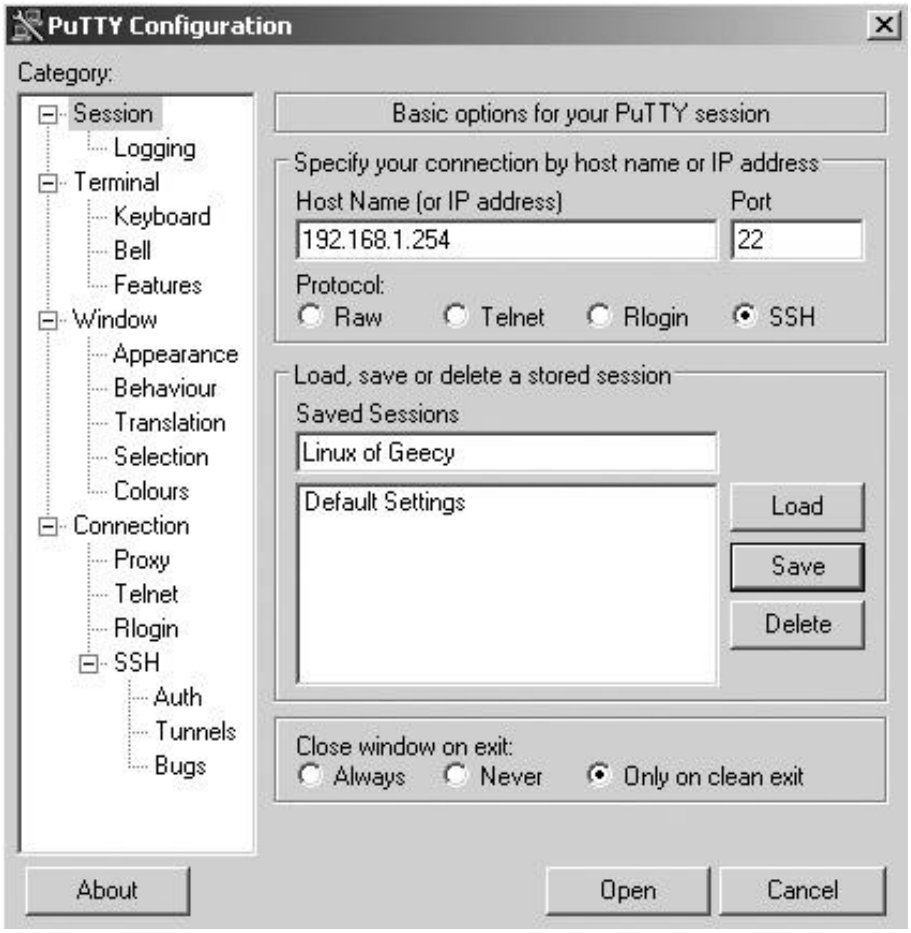

图 1-11

PuTTY 的使用非常简单。只需要在 IP address 中输入需要访问的 SSH 服务器的 IP 地址“192.168.1.254”，然后选择按钮“Open”。

【提示】为了以后的使用方便，也可以保存配置：填入 IP 地址后，再输入一个 Session 名称，例如是“Linux of Geecy”，然后使用“Save”按钮保存配置。以后只需要双击该名称。

如果连接成功，将出现登录提示：

```
login as :
```

然后输入帐号登录。接下来的操作就与直接在控制台的操作一样了。

远程登录使用 Linux 系统比直接在控制台使用要方便，例如可以方便地与 Windows 系统互相复制文本内容、可以同时自由切换于两个不同操作系统等。

1.5 练 习

1、 Cell 宽带数据公司的员工 Tenny 是一位 Linux 爱好者，他决定利用周末时间使用家里一台闲置的个人计算机安装 Debian GNU/Linux 3.0 系统。

2、 Tenny 对 date 命令所输出的时间格式不是很满意，他希望的输出格式是“年-月-日 时:分:秒”，且以 12 小时制表示。例如“2005-01-27 10:14:37 AM”。请借助 man 命令的帮助给出能完成此任务的 date 命令。

【提示】 已知命令“date +%m'-'%d”的输出结果是“01-27”。

*3、Tenny 家里虽然拥有两台计算机主机（一台主机安装有 Microsoft Windows 系统，另一台安装有 Debian GNU/Linux 系统），但是只有一个显示器。为了使用这两种操作系统，他需要频繁地在两台主机之间插拔显示器的数据传输线。现在，Tenny 希望在不用增加显示器的情况下能够同时自由地使用这两个操作系统。请给出解决方案，并具体实施。

【提示】 使用远程登录方式。

*4、Tenny 所在公司的计算机安装的操作系统是 Microsoft Windows 2000。现在，他希望能够利用在公司的空余时间（诸如午休时间等）学习 Linux 系统。请给出解决方案，并具体实施。

【提示】 使用虚拟机技术。

第 2 章　Linux 进阶

本章提要

- Linux 的构成
- Linux 的文件系统
- 外壳程序的使用

2.1　Linux 的构成

Linux 系统由内核、外壳程序、实用程序以及应用程序构成。

2.1.1　内核

内核(Kernel)是 Linux 的核心。内核负责管理和控制计算机资源，分配资源给用户和进程。内核直接与硬件打交道，用户和其他程序都通过内核访问硬件资源，因此使其他程序很容易编写成能够跨平台使用。

查看当前使用的 Linux 发行版本所使用的 Linux 内核，可以执行 kernelversion 命令：

```
ray@geecy:~$ /sbin/kernelversion↙
2.2
```

将显示 Linux 内核的版本号是 2.2。由于普通用户的搜索路径没有包含目录“ /sbin ”，因此该命令需要以绝对路径的形式执行。另一个使用选项“ -a ”(All)的命令 uname 可以显示更详细的信息：

```
ray@geecy:~$ uname -a↙
Linux geecy 2.2.20-idepci #1 Sat Apr 20 12:45:19 EST 2002 i686 unknown
```

2.1.2　外壳程序

Linux 系统提供两种形式的用户接口(UI，User Interface)供用户管理计算机的软件和硬件资源。一种称为命令行接口(CLI，Command Line Interface)，另一种称为图形用户接口(GUI，Graphics User Interface)。

使用命令行接口管理 Linux 系统是最基本和最重要的方式。Linux 系统使用外壳程序(Shell)实现用户与系统之间的命令交互，即用户在命令提示符(Command Prompt)下输入键盘命令，然后外壳程序接收、分析和执行该命令，最后用户在屏幕上得到反馈结果信息。不管命令执行结果成功与否，外壳程序总是再次给出命令提示符，等待接收用户输入的下一个命令，如此不断循环。用户通过使用外壳程序可以管理计算机的所有资源。

图形用户接口在使用上有一定的直观性优势，但到目前为止，很多重要的任务依然必须

由命令行接口完成。就算是相同的任务，如果由命令行接口来完成将会比使用图形用户接口要简捷高效很多。

2.1.3 实用程序和应用程序

Linux 的发行版本通常包含了一些常用的工具性的实用程序(Utility)，供普通用户的日常操作和管理员的维护操作。此外,Linux 系统还有成百上千的第三方应用程序(Application)可供选用，例如数据库管理系统、文字处理系统、Web 服务器程序等。

发行版本 Debian GNU/Linux 3.0 就包含的软件包多达 8710 个，可以供用户选择和使用的实用程序和应用程序极其丰富。

2.2 Linux 的文件系统

2.2.1 文件系统格式

Debian GNU/Linux 3.0 发行版本默认使用 2.2 版本的内核，主文件系统格式是 ext2。比较新一些的内核除了支持 ext2 外还支持该格式的新版本，即 ext3。

Linux 系统为了兼容使用其他格式类型的文件系统（例如其他操作系统所使用的 FAT12、FAT16、FAT32、NTFS、MINIX、UFS、ISO9660 等格式），引入了虚拟文件系统(VFS，Virtual File System)。对于能够被 Linux 系统支持的每种文件系统，都要提供给 VFS 一个相同的接口，这样所有的文件系统对于 Linux 内核和其他程序来说看起来都是相同的。VFS 接口层使得用户可以在系统中同时挂载很多不同格式类型的文件系统。

使用 mount 命令可以完成挂载各种文件系统的任务。例如，先放入一张光盘到 CD-ROM，然后使用 mount 命令把光盘挂载到目录“ /cdrom ”下：

```
ray@geecy:~$ mount /cdrom↙
```

以后对目录“ /cdrom ”的操作实际上是对该光盘的操作。例如，使用 ls 命令：

```
ray@geecy:~$ ls /cdrom↙
```

可以查看光盘的内容。关于 ls 命令的使用，请参考“ 3.1.3　列出目录内容 ”。

单独的 mount 命令可以查看 Linux 系统当前已经挂载的各种文件系统。例如：

```
ray@geecy:~$ mount↙
/dev/hda1 on / type ext2 (rw,errors=remount-ro)
proc on /proc type proc (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/cdrom on /cdrom type iso9660 (ro,noexec,nosuid,nodev,user=ray)
```

当不需要使用被挂载的文件系统，用户可以把它卸载。使用 umount 命令实现卸载：

```
ray@geecy:~$ umount /cdrom↙
```

对于已经挂载的光盘，只有在被卸载后才可以从 CD-ROM 中取出。

2.2.2 目录结构

Linux 继承了 Unix 系统的传统目录结构。

这是一个层次结构，最顶层是整个目录结构的根(Root)，或称为根目录，使用符号“ / ”表示。在根目录下一般会设置如下的一些基本目录：

- /bin 和/sbin 目录

  这两个目录存放了很多实用程序。这些实用程序是 Linux 操作系统的一些常用命令。

- /etc 目录

  该目录存放与 Linux 系统相关的文本配置文件，修改这些文件就可订制系统的特性，控制系统的行为。
- /home 目录

  该目录一般存放各个用户的个人目录。
- /dev 目录

  该目录存放所有与设备有关的文件。
- /lib 目录

  该目录存放编译器所需要包含和链接的库文件。
- /var 目录

  该目录存放一些特定的动态信息，例如系统日志文件。

例如使用 ls 命令可以查看根目录下的内容：

```
ray@geecy:~$ ls -F /↙
bin/    dev/     home/    lost+found/  proc/  tmp/  vmlinuz@
boot/   etc/     initrd/  mnt/         root/  usr/
cdrom/  floppy/  lib/     opt/         sbin/  var/
```

关于 ls 命令的使用，请参考“3.1.3　列出目录内容”。

2.2.3　命名规范

在 Linux 中，目录和文件的命名：

- 长度最多是 256 个字符
- 可以包含大写字母和小写字母，并且区分大小写
- 可以包含空格等特殊字符，但必须使用引号
- 不可以包含“/”字符

2.2.4　路径概念

路径(Path)就是系统寻找到某文件或目录所需要经过的一系列目录，它们以符号“/”分隔。分两种形式的路径：

- 绝对路径

  绝对路径总是以“/”开头。例如“/usr/bin”就是绝对路径的例子。
- 相对路径

  用户在使用 Linux 系统的时候，总是处在某个目录下（即执行命令的时候以该目录作为参考位置），该目录称为用户的工作目录（Working Directory）。

  相对路径不以“/”开头。例如“chapter2/answer.txt”就是相对路径的例子，它相对于用户的当前工作目录，假设为 “/home/ray”，那么当用户发出某个命令去操作文件

“chapter2/answer.txt”的时候，Linux 系统真正要操作的文件是“/home/ray/chapter2/answer.txt”

2.2.5 文件类型

除了普通的程序文件和数据文件，Linux 把所有的设备都当作文件看待，读写相应的设备文件就相当于操作设备。

Linux 中有三种类型的文件：

- 一般文件

  大多数日常使用的文件，比如数据文件、程序文件、可执行脚本等都是以一般文件的形式存放。

- 目录文件

  目录文件简称目录，包含该目录下的文件信息。

- 特殊文件

  Linux 系统中存在大量的特殊文件，这当中有很多是与输入输出设备关联的设备特殊文件，一般存放在目录/dev 中。

一般来说，如果没有特别的说明，后面所简称的“目录”就是目录文件，而“文件”则指除目录文件外的其他文件。有时候提到的“文件”可能是指广义上的文件，即包括目录文件，请根据上下文判断。

2.3 外壳程序的使用

2.3.1 外壳程序的种类

Linux 提供的外壳程序（Shell）一般有 Bourne Shell、C Shell、Korn Shell、Bash Shell、Tcsh Shell、A Shell、Z Shell 等。

其中，最常用的是 Bash Shell。它是“Bourne Again Shell”的缩写，是 Bourne Shell 的增强版。大多数的 Linux 发行版本以 Bash Shell 作为默认的外壳程序。

用户可以查看当前使用的 Linux 系统版本所提供的 Shell。例如：

```
ray@geecy:~$ cat /etc/shells↙
```

将显示系统提供的所有的 Shell。cat 命令的使用请参考“3.2.1 显示文件内容”。

对于执行一些日常使用的命令，各个 Shell 没有很大的差别，兼容性很好，一般只是默认的命令提示符而已。只有在执行一些高级任务（诸如设置环境变量等）以及 Shell 脚本设计时，不同 Shell 之间的差异才表现出来。

2.3.2 Bash Shell 的使用

Bash Shell 具有很多优良特性，了解这些特性可以让用户的日常操作得心应手。

■ 命令补全

用户在命令提示符下输入某一路径的时候，可以只输入前面一部分，然后按<Tab>键，如果后面部分内容是唯一的，则 Bash Shell 会自动补全。如果有重复，那么用户可以再按一次<Tab>键，Bash Shell 将显示可选的路径内容。例如，如果用户需要执行命令“/sbin/kernelversion”，在可以在输入了“/sbin/k”后尝试按一下<Tab>键：

```
ray@geecy:~$ /sbin/k<Tab>
```

但没有反应。这时候用户可以再按一次<Tab>键：

```
ray@geecy:~$ /sbin/k<Tab><Tab>
kallsyms        kernelversion  klogd
kbdrate         killall5       ksyms
```

用户可以清楚看到，下一个要输入的字符是 e。然后继续刚才的输入，按 e，再尝试按<Tab>键：

```
ray@geecy:~$ /sbin/ke<Tab>
```

这时候已经没有重复的路径了，Bash Shell 立即自动补全后面的部分，变为：

```
ray@geecy:~$ /sbin/kernelversion↙
```

最后，用户按回车键就可以执行该命令。命令补全特性很大程度地方便了用户的使用，也提高用户的工作效率。因此建议每个用户要充分利用命令补全功能。

■ **命令历史**

Bash Shell 把用户执行过的命令保存在一个历史文件中，如果用户需要重复执行这些曾经使用过的命令，可以使用上下方向键“ ↑ ”和“ ↓ ”选择其中一个，然后按回车键执行。

如果只是找到相近的命令，可以先使用左右方向键“ ← ”和“ → ”对其进行编辑，然后再按回车键执行。例如，先执行：

```
ray@geecy:~$ ls -F ~↙
ray@geecy:~$
```

然后使用方向键“ ↑ ”找到命令，再使用方向键“ ← ”后退增加一个“ a ”,最后按回车键执行执行：

```
ray@geecy:~$ ls -aF ~↙
./  ../  .bash_history  .bash_profile  .bashrc
```

关于 ls 命令的使用，请参考“ 3.1.3　列出目录内容 ”。

■ **搜索路径**

到目前为止，用户在执行命令的时候遇到过两种情况：一是，对有些命令的执行用户只需要输入命令本身的名字就可以；二是，对另一些命令则要求用户输入该命令的绝对路径系统才能识别。

前者的例子很多，例如“ passwd ”、“ date ”以及“ cal ”等；后则也有一些，例如“ /sbin/kernelversion ”。

Bash Shell 使用 PATH 变量预先指定了一些存放有可执行文件的常用目录，如果用户输入要执行的命令只是给出可执行文件的文件名，那么 Bash Shell 就在 PATH 所指定的各个目录中依次查找该命令是否存在。例如，可以使用下面的命令查看自己的 PATH 变量的默认设置：

```
ray@geecy:~$ echo $PATH↙
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
```

关于该命令的使用，请参考“ 7.2.2　变量的读入与输出 ”和“ 7.2.3　系统环境变量 ”。

不管是在 PATH 所指定的各个目录中找不到需要执行命令，还是因直接指定绝对路径而找不到需要执行命令，Bash Shell 都会给出“ 命令找不到 ”的提示信息：

```
ray@geecy:~$ abc↙
bash: abc: command not found
```

2.4 练 习

1、 请借助 man 命令回答下列问题：

(1) w 命令的功能是什么？如何使用？

(2) who 命令的功能是什么？如何使用？

(3) df 命令的功能是什么？如何使用？

(4) top 命令的功能是什么？如何使用？

第 3 章　Linux 文件管理

本章提要

- 目录基本操作
- 文件基本操作
- 文件的链接
- 使用通配符
- 文件的搜索
- 文件的压缩

3.1　目录基本操作

【问题的提出】

Geecy 软件开发公司的开发人员 Tom 将要参与一个新项目的开发。现在他需要在自己的个人目录下面建立一个子目录 project，然后在该新建的子目录下面再分别建立三个子目录：Source、Doc 和 Script。以便以后用来存放开发过程中需要用到的开发文档和源程序等文件。

【问题分析】

用户在使用 Linux 系统的时候，总是处在某个目录下（即执行命令的时候以该目录作为参考位置），该目录称为用户的工作目录（Working Directory）。

在 Linux 系统中，每个用户都有一个个人目录，也称为 HOME 目录，位置一般是在目录“/home”的下面（例如 Tom 的个人目录是“/home/tom”），用来让用户保存自己的数据文件。因此，用户在工作过程中建立的文件一般是以子目录归类并分门别类的保存在该个人目录下，以方便以后的查找和使用。

当用户刚登录进 Linux 系统的时候，系统默认地把该用户的个人目录设定为他当前的工作目录。因此，用户的个人目录也称为起始目录。

用户对目录的基本操作有：列出当前的工作目录、改变当前的工作目录、列出某些目录的内容、创建一些新的目录和删除某些目录等。对于这些操作命令的使用越是熟悉，那么对于用户的日常工作越是高效，往往能够达到事半功倍的效果。

3.1.1　列出当前工作目录

列出当前工作目录(Print Working Directory)，需要使用命令“pwd”。该命令的使用方法很简单，单独使用即可执行。

假设 Tom 登录到 Linux 系统就立即执行 pwd 命令：

```
tom@geecy:~$ pwd↙
/home/tom
```

系统将以绝对路径的形式在屏幕上打印出 Tom 当前所在的工作目录“ /home/tom ”。一般来说，用户在使用 Linux 系统的过程中随时需要查看自己当前所在的工作目录，以便在使用其它命令的时候能够充分享受“ 相对路径 ”所带来的化繁为简的好处。

3.1.2 改变工作目录

很多命令都需要以一个指定的路径为参数才能工作。为了更方便的应用相对路径，可以重新指定一个不同的目录作为当前的工作目录。这称为改变工作目录(Change Directory)。

用户可以使用 cd 命令来改变当前的工作目录。cd 命令可以使用两种类型的参数。

■ 使用绝对路径作为参数：

例如：下面例子先以绝对路经作参数的方式把当前的工作目录修改为“ /etc ”，然后下一行出现的新命令提示符已经由 tom 改变为 etc。最可靠的办法是继续使用一个 pwd 命令进行验证，第 3 行的输出结果表明，该用户的当前工作目录已经是“ /etc ”(根目录下的 etc 目录)。

```
tom@geecy:~$ cd /etc↙
tom@geecy:/etc$ pwd↙
/etc
```

【提示】发行版本 Debian GNU/Linux 3.0r3 的 Bash Shell，其默认的命令提示符已经包含有当前工作目录的绝对路径的显示，对用户的使用比较方便，但是各个 Linux 发行版本的默认配置不一定相同。这里使用 pwd 命令验证，是为了兼容性。

又如：下面例子以绝对路经作参数的方式把当前的工作目录修改为“ /usr/bin ”,同样也有验证。

```
tom@geecy:/etc$ cd /usr/bin↙
tom@geecy:/usr/bin$ pwd↙
/usr/bin
```

■ 使用相对路径作为参数：

“ .. ”代表的意义是当前工作目录的父目录。因此，命令“ cd .. ”把当前工作目录由“ /usr/bin ”改变为其父目录“ /usr ”。

```
tom@geecy:/usr/bin$ pwd↙
/usr/bin
tom@geecy:/usr/bin$ cd ..↙
tom@geecy:/usr$ pwd↙
/usr
```

“ . ”代表的意义是当前工作目录本身。因此，命令“ cd ./sbin ”把当前工作目录由“ /usr ”改变为该目录下面的子目录 sbin，即“ /usr/sbin ”。

```
tom@geecy:/usr$ pwd↙
/usr
tom@geecy:/usr$ cd ./sbin↙
tom@geecy:/usr/sbin$ pwd↙
/usr/sbin
```

由于“.”的特殊性，上面的例子中命令“cd ./sbin”与“cd sbin”具有一样的功能。因此很少使用。

“~”代表的意义是当前登录用户的个人目录。不管用户当前的工作目录是什么，使用命令“cd ~”都会把用户的工作目录设定为该用户的个人目录。例如下面例子，用户 Tom 的当前工作目录是“/usr/bin”，执行命令“cd ~”后其工作目录改变为 Tom 的个人目录，即“/home/tom”。

```
tom@geecy:/usr/sbin$ pwd↙
/usr/sbin
tom@geecy:/usr/sbin$ cd ~↙
tom@geecy:~$ pwd↙
/home/tom
```

在 Linux 系统中，由于用户的个人目录的特殊性，用户一般都会对它频繁访问。因此为简化使用，cd 命令的不带任何参数的形式，其效果与“cd~ ”是一样的。因此单独的命令“cd”与带特殊参数的命令“cd ~”的作用相同。

如果要设定的工作目录就是上次最后所使用的工作目录，可以使用“-”快速后退到上一次所使用的工作目录并将其作为当前的工作目录。例如：

```
tom@geecy:~$ cd /usr/local/share↙
tom@geecy:/usr/local/share$ pwd↙
/usr/local/share
tom@geecy:/usr/local/share$ cd /tmp↙
tom@geecy:/tmp$ pwd↙
/tmp
tom@geecy:/tmp$ cd -↙
tom@geecy:/usr/local/share$ pwd↙
/usr/local/share
```

首先，当前的工作目录设定为“/usr/local/share”，然后又将当前的工作目录设定为“/tmp”。这时候只需要使用命令“cd -”就可以返回到使用上一次的工作目录“/usr/local/share”作为当前的工作目录。

3.1.3 列出目录内容

目录中可以包含有文件和子目录，子目录又可以包含有文件和子目录，如此递归下去。用户要列出(List)某个目录的内容，可以使用 ls 命令。

单独的命令“ ls ”将列出当前工作目录的内容：

```
tom@geecy:~$ cd /usr↙
tom@geecy:/usr$ ls↙
bin   etc    include   lib      local  share  tmp
dict  games  kerberos  libexec  sbin   src    X11R6
```

也可以为 ls 命令指定一个绝对路径或相对路径作为参数，例如：

```
tom@geecy:~$ ls /usr↙
bin   etc    include   lib      local  share  tmp
dict  games  kerberos  libexec  sbin   src    X11R6
tom@geecy:~$ pwd↙
/home/tom
tom@geecy:~$ ls ../../usr↙
bin   etc    include   lib      local  share  tmp
dict  games  kerberos  libexec  sbin   src    X11R6
```

首先，命令“ ls /usr ”以绝对路径“ /usr ”作为参数，列出目录“ /usr ”下的内容；然后，又以相对路径“ ../../usr ”作为参数，再次列出同一目录的内容。

在这个例子中，用户的当前工作目录是“ /home/tom ”，因此“ .. ”代表的目录是“ /home ”，“ ../../ ”代表的目录是根目录“ / ”，所以“ ../../usr ”代表的目录正是根目录下的“ usr ”，即“ /usr ”。

虽然前述的三个 ls 命令都把目录“ /usr ”的内容打印到了屏幕，但是用户无法区分哪些是子目录，哪些是文件。因此，为了让用户能够更精确的控制 ls 命令所输出的内容，ls 命令提供了若干个常用的选项(Option)供用户选用。

使用-F 选项可以让用户区分出所列出的分别是什么类型的内容。改进上面的例子：

```
tom@geecy:~$ cd /usr↙
tom@geecy:/usr$ ls -F↙
bin/   etc/    include/   lib/      local/  share/  tmp@
dict/  games/  kerberos/  libexec/  sbin/   src/    X11R6/
```

由此用户可以清晰看出，所列出的内容中，名字后面带有“ / ”的表示是一个目录，而带有“ @ ”的则表示是一个符号链接文件(Symbolic Link File)。

又例如下面的例子：

```
tom@geecy:~$ ls -F /bin↙
arch*        df*            hostname*  netstat*        sort*
ash*         dmesg*         igawk*     nice*           stty*
ash.static*  dnsdomainname@  ipcalc*    nisdomainname@  su*
(省略后面内容)
```

以“ * ”为结尾显示的文件表示是一个可执行文件(Executable File)。

需要注意，这些附加的“ / ”、“ * ”和“ @ ”等并不是名字的一部分。

另一个很常用的选项是“ -l ”，它使用长(Long)列表的格式显示目录的内容，例如：

```
tom@geecy:~$ ls -l /bin↙
total 4816
-rwxr-xr-x    1 root     root         2644 Feb 24  2003 arch
-rwxr-xr-x    1 root     root        92444 Feb  6  2003 ash
-rwxr-xr-x    1 root     root       492968 Feb  6  2003 ash.static
lrwxrwxrwx    1 root     root            4 Jan  8 07:43 awk -> gawk
-rwxr-xr-x    1 root     root        10848 Feb 18  2003 basename
-rwxr-xr-x    1 root     root       626028 Feb 11  2003 bash
lrwxrwxrwx    1 root     root            4 Jan  8 07:41 bash2 -> bash
lrwxrwxrwx    1 root     root            3 Jan  8 07:43 bsh -> ash
-rwxr-xr-x    1 root     root        14364 Feb 18  2003 cat
(省略后面内容)
```

命令结果是先给出一个统计信息，以 KB(1024 字节)为单位指出该目录所占用的磁盘空间的大小（例如这里所示的 4816KB），然后以 9 列的形式显示，用户将获取更多的有用信息：

第 1 列：文件类型(File Type)和文件访问权限(FAP，File Access Permission)；

第 2 列：文件链接数。

第 3 列：文件拥有者。

第 4 列：组拥有者。

第 5 列：文件大小，单位是字节(Byte)。

第 6-8 列：文件最后一次修改的日期和时间。

第 9 列：文件名称。

第 1 列的内容（如“-rwxr-xr-x”）共有 10 个字符，其中左边第 1 个字符（如“-”）指出文件的类型：

“-”指出该文件是普通文件，例如这里的 “arch”、“ash”和“bash”等文件就是普通文件；

“l”指出该文件是链接文件，例如这里的 “awk”、“bash2”和“bsh”等文件就是链接文件，它们分别指向文件“gawk”、“bash”和“ash”。

常见的文件类型的代表符号说明如表 3-1 所示。

表 3-1

符　号	文 件 类 型
-	普通文件
d	目录(Directory)
b	块(Block)特殊文件
c	字符(Character)特殊文件
l	符号链接(Symbolic Link)文件

剩下的 9 个字符（如“rwxr-xr-x”）描述了该文件的访问权限的设置。关于文件的访问

权限、文件拥有者和组拥有者，请参考“4.2　文件权限管理”。

又例如下面的例子：

```
tom@geecy:~$ ls -l /usr↙
total 56
drwxr-xr-x    2 root     root         8192 Jan  8 07:57 bin
drwxr-xr-x    2 root     root         4096 Jan 24  2003 dict
drwxr-xr-x    2 root     root         4096 Jan 24  2003 etc
drwxr-xr-x    2 root     root         4096 Jan 24  2003 games
drwxr-xr-x    3 root     root         4096 Jan  8 07:57 include
drwxr-xr-x    4 root     root         4096 Jan  8 07:43 kerberos
drwxr-xr-x   22 root     root         4096 Jan  8 07:57 lib
drwxr-xr-x    4 root     root         4096 Jan  8 07:51 libexec
drwxr-xr-x   11 root     root         4096 Jan  8 07:39 local
drwxr-xr-x    2 root     root         4096 Jan  8 07:57 sbin
drwxr-xr-x   35 root     root         4096 Jan  8 07:57 share
drwxr-xr-x    3 root     root         4096 Jan  8 07:39 src
lrwxrwxrwx    1 root     root           10 Jan  8 07:39 tmp -> ../var/tmp
drwxr-xr-x    7 root     root         4096 Jan  8 07:39 X11R6
```

第 1 行给出的统计信息是目录“/usr”所占用的磁盘空间是 56KB。子目录里面内容的大小是没有纳入这个统计的。每个子目录本身（即不包括其下的内容）只占用 4KB 的磁盘空间。因此这里是 4KB × 14 = 56KB。

同样，第 1 列中的第 1 个字符指出了文件类型：

“d”表示是一个目录（目录是一个特殊的文件）。例如这里的“bin”、“lib”和“src”等就是目录“/usr”下的子目录；

链接文件既可以指向一个文件，也可以指向一个目录。例如这里的“tmp”就是一个指向目录“../var/tmp”(即绝对路径“/var/tmp”)的链接文件。

选项还可以组合使用，例如，命令“ls -lF /usr”或“ls -l -F /usr”将同时体现两种选项的效果。

除了选项“-F”和“-l”，还有一些常用的选项，其作用的说明列于表 3-2。

表 3-2

选　项	作　用
-a	显示所有(All)的目录和文件，包括隐藏的目录和文件（以点“.”开头的）
-l	以长(Long)格式显示结果

-r	以逆向(Reverse)排序的次序显示
-t	以文件的最后修改时间(Time)排序显示
-A	显示几乎所有(Almost all)的目录和文件（“.”和“..”除外）
-F	显示目录和文件的名称，并给出文件类型
-R	以递归(Recursive)方式显示该目录的内容和所有子目录的内容
-S	以文件大小(Size)的递降次序排序显示

请使用命令“man ls”获取详细的帮助。

3.1.4 创建新目录

如果在一个单独的目录中存放成百上千个文件，那么用户要从中找出所需要的文件会花费很长时间，也不便于对文件的管理。因此建议用户适当的在各个层次创建目录(Make Directory)，分门别类的对大量的文件进行轻松管理。

建立目录可以使用 mkdir 命令。

由于 Linux 系统的默认安全设置，普通用户要创建目录或文件，一般只能创建在用户的个人目录和临时目录“/tmp”这两个地方。

下面例子首先使用不带参数的命令“cd”把当前工作目录切换到用户的个人目录，然后使用以相对路径“language”为参数的命令“mkdir language”在当前工作目录下建立一个名为“language”的子目录（即绝对路径为“/home/tom/language”）。

```
tom@geecy:~$ cd↙
tom@geecy:~$ pwd↙
/home/tom
tom@geecy:~$ mkdir language↙
tom@geecy:~$ ls -F↙
language/
```

用户也可以使用绝对路径作为参数。例如：

```
tom@geecy:~$ mkdir /tmp/public↙
tom@geecy:~$ ls -F /tmp↙
public/
```

本例子使用绝对路径“/tmp/public”作参数，在目录“/tmp”下建立一个名为“public”的子目录（绝对路径就是所使用的参数“/tmp/public”）。

如果要连续建立几个目录，不必繁琐的输入几次命令，只需要把这些待建立的目录名称以空格为分隔作为参数，使用一次 mkdir 命令即可。例如：

```
tom@geecy:~$ cd↙
tom@geecy:~$ pwd↙
/home/tom
```

```
tom@geecy:~$ ls -F↙
language/
tom@geecy:~$ mkdir document language/cpp language/java↙
tom@geecy:~$ ls -F↙
document/  language/
tom@geecy:~$ ls -F language↙
cpp/  java/
```

将一次性的在目录“/home/tom”下建立子目录“document”(即“/home/tom/document”)并且在目录“/home/tom/language”下建立两个子目录“cpp”和“java”(即分别为“/home/tom/language/cpp”和“/home/tom/language/java”)。

如果需要创建的目录名称包含空格等特殊字符，用户可以使用双引号来实现。例如：

```
tom@geecy:~$ cd ~/language↙
tom@geecy:~$ mkdir "C++ Program"↙
tom@geecy:~$ ls -F↙
cpp/  C++ Program/  java/
```

如果需要一次垂直地建立多个层次的目录结构，且每一层次都只有一个子目录，那么可以使用一个选项“-p”实现，该选项能够自动判断建立其所需要的父目录(Parent)。例如：

```
tom@geecy:~$ cd↙
tom@geecy:~$ mkdir -p aa/bb/cc/dd↙
tom@geecy:~$ ls -F↙
aa/  document/  language/
tom@geecy:~$ ls -F aa/bb/cc↙
dd/
```

将在用户的个人用户下建立一个深层次的子目录“dd”。

如果用户的个人用户下还没有子目录“aa”，则自动建立；如果目录“aa”下还没有子目录“bb”，则自动建立；如果目录“bb”下还没有子目录“cc”，则自动建立；如果目录“cc”下还没有子目录“dd”，则自动建立。其效果相当于依次发出四个命令：

```
tom@geecy:~$ mkdir aa↙
tom@geecy:~$ mkdir aa/bb↙
tom@geecy:~$ mkdir aa/bb/cc↙
tom@geecy:~$ mkdir aa/bb/cc/dd↙
```

3.1.5 删除目录

某些不再需要的目录，如果它里面的内容已经为空（即不含有任何子目录或文件），则可以删除目录(Remove Directory)。用户可以使用 rmdir 命令实现。

对于前面所建立的目录“aa”，由于里面还有子目录，即目录“aa”的内容不为空，用户不能直接删除。如果一定需要删除，那么可以从最深层次的目录“dd”开始，逐个删除。当然，这里假设这些目录下都没有文件。

```
tom@geecy:~$ cd↙
tom@geecy:~$ ls -F aa/bb/cc/dd↙  [验证目录“dd”存在且内容为空]
tom@geecy:~$ rmdir aa/bb/cc/dd↙
tom@geecy:~$ ls -F aa/bb/cc↙     [验证目录“cc”存在且内容为空]
tom@geecy:~$ rmdir aa/bb/cc↙
tom@geecy:~$ ls -F aa/bb↙        [验证目录“bb”存在且内容为空]
tom@geecy:~$ rmdir aa/bb↙
tom@geecy:~$ ls -F aa↙           [验证目录“aa”存在且内容为空]
tom@geecy:~$ rmdir aa↙
tom@geecy:~$ ls -F↙
document/  language/
```

跟建立目录类似，也可以一次删除多个空目录，例如：

```
tom@geecy:~$ cd↙
tom@geecy:~$ rmdir document /tmp/public↙
tom@geecy:~$ ls -F /tmp↙
tom@geecy:~$ ls -F↙
language/
```

这里将一次删除“/home/tom/document”和“/tmp/public”两个空的目录。

如果将要删除的目录包含空格等特殊字符，则需要使用双引号。例如：

```
tom@geecy:~$ cd↙
tom@geecy:~$ ls –F language↙
cpp/  C++ Program/  java/
tom@geecy:~$ rmdir "language/C++ Program"↙
tom@geecy:~$ ls -F language↙
cpp/  java/
```

这将删除名字带有空格的空目录“C++ Program”。当然，用户也可以先进入目录“language”，然后再发出删除命令。例如：

```
tom@geecy:~$ cd ~language↙
tom@geecy:~/language$ ls –F↙
cpp/  C++ Program/  java/
tom@geecy:~/language$ rmdir " C++ Program"↙
tom@geecy:~/language$ ls -F↙
cpp/  java/
```

将具有同样的效果。

如果需要删除的目录包含有文件，那么应该怎么办？请参考“3.2.4　移动文件”。

【解决方案】

现在，Tom 已经能够自由的使用这些对目录进行基本操作的命令了。但他还是小心翼翼，每一个命令操作后，都使用“pwd”、“ls”等命令进行验证，检验操作是否真正成功，毕竟是初次使用嘛。

```
tom@geecy:~$ cd↙
tom@geecy:~$ pwd↙
/home/tom
tom@geecy:~$ mkdir project↙
tom@geecy:~$ ls -F↙
project/
tom@geecy:~$ cd project↙
tom@geecy:~/project$ pwd↙
/home/tom/project
tom@geecy:~/project$ mkdir Source Doc Script↙
tom@geecy:~/project$ ls -F↙
Doc/  Script/  Source/
```

Tom 首先使用不带参数的 cd 命令，把当前工作目录设定为自己的个人目录，再使用 pwd 命令加以验证。

然后使用 mkdir 命令在当前工作目录下建立名为“project”的子目录，并使用 ls 命令验证。

接着，使用带参数的 cd 命令，把当前工作目录设定为新建的子目录“project”，并使用 pwd 命令加以验证。

最后，使用 mkdir 命令在当前工作目录下一次建立三个子目录“Source”、“Doc”和“Script”，并使用 ls 命令验证。

任务完成，问题解决。

后来，Tom 跟同事交流使用经验，得知 Linux 系统的一个重要特点：没有（出错）消息就是好消息。也就是说，对于大多数的命令来说，如果用户执行命令之后没有得到一个出错信息的提示，那么就表明该命令已经按照用户的意图执行成功。

于是 Tom 改进了上面操作，把一些可有可无的验证命令省略去，简化为：

```
tom@geecy:~$ cd↙
tom@geecy:~$ mkdir project↙
tom@geecy:~$ mkdir project/Source project/Doc project/Script↙
tom@geecy:~$ ls -F project↙
Doc/  Script/  Source/
```

当然，在执行上面这些命令之前，必须先执行下面的删除命令把前面刚建立的 4 个空目录删除：

```
tom@geecy:~$ cd ~/project↙
```

```
tom@geecy:~/project$ rmdir Source Doc Script↙
tom@geecy:~/project$ cd ..↙
tom@geecy:~$ rmdir project↙
```

3.2 文件基本操作

【问题的提出】

Linux 系统的帐号资料（诸如用户名、密码等）是以文本文件的形式保存在/etc 目录下，这些文件在整个系统中具有举足轻重的地位。

Geecy 软件开发公司的 Linux 系统管理员 Ray 的日常工作之一是需要定期的查看系统的帐号文件，分析是否有异常情况。如果正常，就备份该密码文件；如果不正常，就把以前备份的正常的密码文件恢复出来，以供系统正常工作。

【问题分析】

Linux 系统的帐号文件由三个文件组成：

- /etc/passwd
- /etc/group
- /etc/shadow

这三个文件都是文本文件，都以行为单位保存用户的信息。文件“ /etc/passwd ”保存有用户名、个人目录等资料，但没有密码。在 Unix 发展的早期，此文件是包含有密码的，但后来由于安全问题，很多 Unix 版本都不把密码放在此文件了，而通常会把密码加密后存放于另外的文件中。Linux 系统是把密码加密后存放于文件“ /etc/shadow ”中。文件“ /etc/group ”则存放有用户组的信息，包括组名、组的成员等。关于用户和组的进一步介绍，请参考“ 4.1 用户管理 ”。

对于文件“ /etc/passwd ”和“ /etc/group ”，Linux 系统的默认权限设置是，普通用户只有读的权限（只能查看而不能修改内容），而对于文件“ /etc/shadow ”，对普通用户来说连查看的权限都没有（复制就更加不行了）。

因此，如果需要完整地备份这三个帐号文件，用户就需要使用 su 命令以系统管理员的权限登录去执行备份操作。

Ray 所要做的日常工作，通常需要涉及到显示文件内容、复制文件、删除文件、移动文件等基本的文件操作。当然，普通的程序开发人员在 Linux 系统中的日常的开发工作，同样也要经常涉及到这些基本操作。对于这些操作命令的使用越是熟悉，那么对于用户的日常工作越是高效，往往能够达到事半功倍的效果。

3.2.1 显示文件内容

Linux 继承了 Unix 的风格，采用文本文件(Text File)对系统进行配置管理（例如帐号管理）。要了解系统的配置，用户需要在屏幕上显示这些配置文件的内容。对于用户的数据文件（例如开发人员的源程序），通常也是采用文本文件保存，用户也需要在屏幕上显示数据文件的内容。

常用的显示文件内容的命令有 cat、more 和 less。

下面例子使用 cat 命令显示文本文件“ /etc/hosts ”的内容：

```
ray@geecy:~$ cat /etc/hosts↙
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               geecy localhost.localdomain localhost
```

该文件是 Linux 系统中的一个简单的配置文件，默认的文件内容也比较少（例如 3 行）。再来看一个内容比较长的文本文件：

```
ray@geecy:~$ cat /etc/inittab↙
```

这个文件的内容将在屏幕上一闪而过，滚动而上，最后只剩下文件的最后面部分内容静止在屏幕上。换句话说，用户每次执行此命令都将只能看到同一部分内容（最后那部分）。那么如何才能够完整的查看这些内容比较多的文本文件呢？

如果用户是在 Linux 系统的控制台登录使用系统，那么可以利用<Shift> + <PageUp>和<Shift> + <PageDown>这两个组合键来分别实现屏幕内容的向上和向下的滚动，从而实现能够看到那些被滚动上去的内容。

如果用户是使用仿真终端软件（例如 PuTTY）通过网络登录使用系统，那么可以使用鼠标上下拖动这些软件本身所提供的垂直滚动条来实现对被滚动上去的内容的查看。

当然，还会有更方便的方法。对于长度超过一个屏幕行数的文本文件，cat 命令显得力不从心。这时候用户可以选用 more 命令或 less 命令来轻松完成。

例如：

```
ray@geecy:~$ more /etc/inittab↙
```

由于文件“ /etc/inittab ”的内容长度超过一个屏幕的行数，more 命令从该文件的开头开始，每次以一个屏幕作为一页显示，按空格键可以实现翻屏，显示下一页内容。如果不希望一次翻一页，也可以按回车键，则每次向下滚动一行的内容。

more 命令是单向翻页的，不支持向上一页翻页。如果翻到了最后一页，则自动退出，返回到 Linux 的命令提示符。如果还没有到达最后一页，可以按“ q ”键(Quit)退出。

less 命令是 more 命令的加强版本（比较有趣的是，它们的名字从英文意义上是相对的）。例如：

```
ray@geecy:~$ less /etc/inittab↙
```

less 命令能实现 more 命令的功能，使用更灵活，功能更强大。但在某些 Unix 版本或某些 Linux 发行版本，比如一些嵌入式系统等，为了节约文件系统的空间，没有安装 less 命令，而一般只有基本的 more 命令供用户使用。

除了可以用空格键实现单向的翻屏，less 还可以使用“ ↑ ”（向上方向键）和“ ↓ ”（向下方向键）实现既可以向上也可以向下的逐行滚动。无论是到达最后一页或最后一行，还是在中途，用户都需要按“ q ”键(Quit)退出，返回到 Linux 的命令提示符。

一般只对文本文件进行查看。如果用户查看一个二进制文件，例如：

```
ray@geecy:~$ cat /bin/ls↙
```

如果用户在 Linux 系统的控制台登录使用此命令，屏幕将显示一些混乱的符号。如果用

户是使用仿真终端软件（例如 PuTTY）通过网络登录使用此命令，屏幕也会显示一些混乱的符号，但有时候其副作用会对仿真终端软件产生不良影响，严重的就会令仿真终端软件停止响应，此时必须强行终止仿真终端软件然后重新运行和登录才能继续工作。

more 命令则会自动判断给定的文件，如果是二进制文件，则给出提示信息并停止。例如：

```
ray@geecy:~$ more /bin/ls↙
******** /bin/ls: Not a text file ********
```

less 命令遇到二进制文件，则会向用户给出提示信息并询问是否继续显示（按“y”表示要继续显示)。例如：

```
ray@geecy:~$ less /bin/ls↙
"/bin/ls" may be a binary file.  See it anyway?
```

3.2.2 复制文件

对文件的复制(Copy)操作会产生一个内容一样的副本，通常对其进行进一步的修改或作为备份用途。文本文件和二进制文件都可以被复制。

复制文件可以使用 cp 命令。

先看一个例子：

```
ray@geecy:~$ cp /etc/hosts ~↙
ray@geecy:~$ ls -l ~↙
total 4
-rw-r--r--    1 hdf      hdf           153 Jan 26 08:30 hosts
```

命令 cp 需要两个参数。第一个参数指定将要被复制的原文件的路径，第二个参数指定将要复制到的目的路径，这些路径可以是绝对路径，也可以是相对路径。

因此，本例子的作用是，把文件“/etc/hosts”复制到用户的个人目录下，并且不修改文件名称。可以使用 ls 命令对复制的结果进行检验。

需要注意，如果在复制前，目标路径已经存在同名的文件，则系统会静悄悄的把该同名文件覆盖，不会给出任何的提示信息。

如果要复制到的目的路径就是用户当前的工作目录，那么可以简化为：

```
ray@geecy:~$ cp /etc/passwd .↙
ray@geecy:~$ ls -l ~↙
total 8
-rw-r--r--    1 hdf      hdf           153 Jan 26 08:52 hosts
-rw-r--r--    1 hdf      hdf          1194 Jan 26 08:55 passwd
```

很多时候需要对复制出来的副本加以区别，因此需要在复制的同时把文件名修改。例如：

```
ray@geecy:~$ cd↙
ray@geecy:~$ cp /etc/passwd ./passwd_050125↙
```

```
ray@geecy:~$ cp /etc/passwd ~/passwd_050126↙
ray@geecy:~$ ls↙
hosts  passwd  passwd_050125  passwd_050126
```

只需要在目的路径中指定一个新的文件名（例如“passwd_050125”）就可以实现复制的同时修改文件名。当然，这不会影响原来的文件。

cp 命令提供一个选项“-R”可以递归(Recursive)方式实现整个目录的内容复制。例如：

```
ray@geecy:~$ cp -R /bin /tmp↙
ray@geecy:~$ ls -F /tmp↙
bin/
ray@geecy:~$ ls /tmp/bin↙
arch        df             hostname  netstat          sort
ash         dmesg          igawk     nice             stty
ash.static  dnsdomainname@ ipcalc    nisdomainname@   su
(省略后面内容)
```

将把目录“/bin”以及其下的所有内容（包括所有的文件和子目录）复制到目录“/tmp”下。与单个文件的复制类似，在复制中也可以同时改名。例如：

```
ray@geecy:~$ cp -R /bin ~/bin_bak↙
ray@geecy:~$ ls ~↙
bin_bak/
ray@geecy:~$ ls ~/bin_bak↙
arch        df             hostname  netstat          sort
ash         dmesg          igawk     nice             stty
ash.static  dnsdomainname@ ipcalc    nisdomainname@   su
(省略后面内容)
```

将把目录“/bin”以及其下的所有内容（包括所有的文件和子目录）复制到用户的个人目录下，并把 bin 改名为 bin_bak。

需要注意，如果在复制前，用户的个人目录下已经存在名为“bin_bak”的目录，那么就不是修改名字，而仅仅是复制整个目录“/bin”到目录“~/bin_bak”下而已。

3.2.3 删除文件

对于一些打算以后都不会用到的文件，用户可以删除(Remove)这些文件，腾出更多的磁盘空间以供使用。

删除文件可以使用 rm 命令。

例如：

```
ray@geecy:~$ cd↙
ray@geecy:~$ ls↙
```

```
hosts  passwd  passwd_050125  passwd_050126
ray@geecy:~$ rm passwd_050125↙
ray@geecy:~$ ls↙
hosts  passwd  passwd_050126
```

将把前面例子中复制到用户个人目录下的文件“passwd_050125”从系统中删除。可以使用 ls 命令加以验证。

同样可以一次删除多个文件，只需要把这些待删除的文件名以空格分隔作为参数使用 rm 命令。例如：

```
ray@geecy:~$ ls↙
hosts  passwd  passwd_050126
ray@geecy:~$ rm hosts passwd↙
ray@geecy:~$ ls↙
passwd_050126
```

将把前面例子中复制到用户个人目录下的其中两个文件“hosts”和“passwd”一次删除。

与复制文件类似，rm 命令也提供选项“-R”，让用户可以递归(Recursive)方式一次删除整个指定的目录以及其下的所有内容（包括所有的文件和子目录）。例如：

```
ray@geecy:~$ ls -F /tmp↙
bin/
ray@geecy:~$ rm -R /tmp/bin↙
rm: remove write-protected regular file `/tmp/bin/ps'? y
ray@geecy:~$ ls -F /tmp↙
```

将把前面例子中复制到目录“/tmp”下的“bin”目录整个删除。有些文件可以需要用户使用“y”确认才能删除。这些需要用户确认的文件如果数量比较少，那么用户按几次键盘倒无所谓。但是如果数量是成百上千呢？

rm 命令还提供一个选项“-f”，如果系统需要用户输入 y 或 n 来确认是否真的要删除文件，则不需要用户回答，默认用 y。即是强制(Force)删除的意思。例如：

```
ray@geecy:~$ ls -F ~↙
bin_bak/
ray@geecy:~$ rm -Rf ~/bin_bak↙
ray@geecy:~$ ls -F ~↙
```

将把前面例子中复制到用户个人目录下的“bin_bak”目录整个强制删除，无需用户确认。当然，删除命令也可以写成“rm -R -f ~/bin_bak”，即选项可以分开，也可以合并。

3.2.4 移动文件

如果用户在创建文件的时候，把它放错了位置，或者是在复制文件的时候目的路径搞错了，那么就需要移动(Move)文件到适当的位置，以方便对文件的管理。当然，用户也可以把

该文件复制到正确的位置，然后把原来的文件删除，但是这样做通常会产生引起磁盘文件系统的碎片(Flagment)问题，从而降低了文件系统的读写性能，因此通常不建议这样做。

移动文件可以使用 mv 命令。

例如：

```
ray@geecy:~$ cp /etc/passwd /tmp↙
ray@geecy:~$ ls /tmp↙
passwd
ray@geecy:~$ mv /tmp/passwd ~↙
ray@geecy:~$ ls /tmp↙
ray@geecy:~$ ls ~↙
passwd
```

首先把文件“ /etc/passwd ”复制到目录“ /tmp ”下作为示例。与 cp 命令类似，mv 命令的第一个参数指出待移动的文件的路径，第二个参数指出要移动的目标路径。因此，本例中是把文件“ /tmp/passwd ”由目录“ /tmp ”移动到用户的个人目录下，文件名称保持不变。

除了单个文件的移动，mv 也可以支持对整个目录的移动。例如：

```
ray@geecy:~$ cp -R /bin /tmp↙
ray@geecy:~$ ls -F /tmp↙
bin/
ray@geecy:~$ mv /tmp/bin ~↙
ray@geecy:~$ ls /tmp↙
ray@geecy:~$ ls -F ~↙
bin/    passwd
```

首先把整个目录“ /bin ”复制到目录“ /tmp ”下作为示例。mv 命令可以自动判断指定的待移动路径是文件还是目录，而无需像 cp 命令一样需要额外的选项。因此对整个目录的移动与对单个文件的移动在命令形式上是相同的。

与 cp 命令类似，mv 命令也可以在移动过程中改名。例如：

```
ray@geecy:~$ cp -R /bin /tmp↙
ray@geecy:~$ ls -F /tmp↙
bin/
ray@geecy:~$ mv /tmp/bin ~/bin2↙
ray@geecy:~$ ls /tmp↙
ray@geecy:~$ ls -F ~↙
bin/    bin2/   passwd
```

再次把整个目录“ /bin ”复制到目录“ /tmp ”下作为示例。在 mv 命令的第二个参数中指定目录名称由“ bin ”修改为“ bin2 ”。

Linux 系统没有专门的只是修改文件名或目录名的命令。但根据 mv 命令的特点，可以借

助 mv 命令实现重命名(Rename)的功能。例如：

```
ray@geecy:~$ ls -F ~↙
bin/    bin2/   passwd
ray@geecy:~$ mv passwd passwd_new↙
ray@geecy:~$ ls -F ~↙
bin/    bin2/   passwd_new
ray@geecy:~$ mv bin bin_new↙
ray@geecy:~$ ls -F ~↙
bin2/   bin_new/    passwd_new
```

首先把用户的个人目录下的文件“passwd” 重命名为“passwd_new”，然后把用户的个人目录下的子目录“bin” 重命名为“bin_new”。目录的重命名对其下的内容（包括所有的文件和子目录）并不会产生任何的影响。

【解决方案】

Linux 系统的默认权限设置，对文件“/etc/shadow”的查看都需要系统管理员权限才能实现。因此，Ray 以普通用户权限的帐号（例如“ray”）登录后，使用带选项“-l”的 su 命令临时以系统管理员身份登录，完成管理任务后尽快退出返回到普通用户状态，以减少一些不必要的误操作而引起的对系统的破坏的可能性。

```
ray@geecy:~$ su -↙
Password:
geecy:~# pwd↙
/root
geecy:~# mkdir account_050126↙
geecy:~/account_050126# cd account_050126↙
geecy:~/account_050126# cp /etc/passwd .↙
geecy:~/account_050126# cp /etc/group .↙
geecy:~/account_050126# cp /etc/shadow .↙
geecy:~/account_050126# ls↙
group   passwd  shadow
```

为避免误操作对帐号文件的破坏，Ray 首先建立目录 account_050126，然后备份三个帐号文件到该目录。对帐号文件的查看和分析只需要对该备份文件进行操作即可：

```
geecy:~/account_050126# less passwd↙
geecy:~/account_050126# less group↙
geecy:~/account_050126# less shadow↙
```

使用 less 命令进行查看会比较方便。至于看到内容后如何分析，请参考“4.1　用户管理”。

管理任务执行完毕后，Ray 立即使用 exit 命令退出系统管理员用户(root)的登录：

```
geecy:~/account_050126# exit↙
ray@geecy:~$
```

3.3 文件操作进阶

【问题的提出】

系统配置文件一般是分门别类地存放在目录“/etc”下以及该目录的各级子目录下，但具体的存放位置有时候又因为发行版本的不同而有一些小小的差异。

Geecy 软件开发公司的 Linux 系统管理员 Ray 经常需要查看和修改目录“/etc”下的系统配置文件。如何快速浏览目录“/etc”下的所有子目录的结构和分布，便成为 Ray 当前最迫切需要解决的问题。

【问题分析】

Ray 现在可以使用的方法是：

```
ray@geecy:~$ ls -lR /etc↙
```

即直接使用 ls 命令的两个选项“-l”和“-R”，以长格式、递归方式显示目录“/etc”下的所有文件和子目录。但由于“/etc”下的文件和子目录数量众多，并且显示的内容夹杂大量的文件，因此不利于对其子目录的结构和分布的了解。

比较理想的解决方式是使用一个名为“tree”的命令。该命令能够以树形的方式浏览目录的结构（不会夹杂有文件）。但由于 tree 命令不一定在所有的发行版本都默认安装。因此通用性不是很好。

事实上，对于文件和目录的一些比较复杂的搜索操作，可以灵活应用最基本的通配符和搜索命令 find 实现。

3.3.1 使用通配符

在使用 Linux 系统的时候，用户经常需要查找一些名字具有某些特征的文件。Shell 通常提供通配符(Wildcard Character)，让用户快速找到所需的文件。通配符对于目录名和文件名都适用。

常用的通配符有三种：“*”、“?”和“[ ]”。

■ **使用“*”查找文件**

符号“*”能够匹配文件名中的任意长度的字符串，即能够代表零个、一个或者一个以上的字符。例如：

```
ray@geecy:~$ ls /etc/host*↙
/etc/host.conf  /etc/hostname  /etc/hosts  /etc/hosts.allow  /etc/hosts.deny
```

host*表示匹配以 host 开头的、后面可以连接零个、一个或一个以上的任何字符的文件名。又例如：

```
ray@geecy:~$ ls /bin/*sh↙
/bin/bash  /bin/csh  /bin/rbash  /bin/sh  /bin/tcsh
```

*sh 表示匹配以 sh 结尾的、前面可以连接零个、一个或一个以上的任何字符的文件名。

通配符也可以应用于目录名：

```
ray@geecy:~$ cp -R /*bin .↙
ray@geecy:~$ ls -F↙
/bin    /sbin
```

/*bin 可以匹配根目录下以 bin 结尾的目录名。因此，命令“cp -R /*bin .”相当于下面两个命令的作用：

```
ray@geecy:~$ cp -R /bin .↙
ray@geecy:~$ cp -R /sbin .↙
```

■ 使用“?”查找文件

符号“?”能够匹配文件名中的任意一个字符。

例如：

```
ray@geecy:~$ ls /bin/?sh↙
/bin/csh
ray@geecy:~$ ls /bin/??sh↙
/bin/bash  /bin/tcsh
```

?sh 表示能够匹配以某一个字符开头的、后面是 sh 的共有 3 个字符的文件名。??sh 表示能够匹配以某两个字符开头的、后面是 sh 的共有 4 个字符的文件名。

通配符“*”和“ ？”的使用形式比较类似，差别只是所能匹配的字符在数量上的不同。

■ 使用“[ ]”查找文件

符号“[ ]”能够匹配方括号中的任意一个字符。

例如：

```
ray@geecy:~$ ls /bin/s[abhuw]↙
/bin/sh  /bin/su
```

可以与“s[abhuw]”匹配的字符串有“sa”、“sb”、“sh”、“su”以及“sw”。由于目录/bin 下只存在“sh”和“su”两个文件，因此只列出着两个文件。

符号“[ ]”里面可以使用范围连接符号“-”，例如：

```
ray@geecy:~$ ls /bin/s[a-z]↙
/bin/sh  /bin/su
```

那么“s[a-z]” 可以匹配以 s 开头、第二个字符是小写字母的长度为 2 的字符串。

在实际使用过程中，用户可以根据需要组合使用这些通配符。

3.3.2 文件的搜索

如果需要在某一目录及其所有的子目录中快速搜索具有某些特征的目录或文件，那么可以使用 find 命令。

使用 find 命令的常用语法是：

```
find [Directory] [Option] [-exec Command]
```

其中：

（1）中括号表示是可选部分。

（2）Directory 表示文件名，绝对路径或相对路径都有效。如果省略，则默认是当前工作目录。

（3）Option 表示选项，常用的选项有“ -name ”和“ -type ”等。

（4）选项“ –exec ”指出把搜索结果的目录或文件名传送给命令 Command 作为参数，并逐一运行带参数的命令 Command。

选项“ -name ”指出需要按照目录或文件的名字搜索。例如：

```
ray@geecy:~$ find /bin -name "*sh"↙
/bin/bash
/bin/rbash
/bin/sh
/bin/tcsh
/bin/csh
```

其功能是在目录/bin 下查找名字与"*sh"匹配的文件。搜索结果默认是输出到屏幕。即相当于下面的命令：

```
ray@geecy:~$ ls /bin/*sh↙
```

但 find 命令可以实现的功能要强大得多。例如，如果用户只是知道外壳程序 tcsh 是在目录/usr 下的某一个子目录，那么使用 find 命令可以实现快速定位：

```
ray@geecy:~$ find /usr -name "tcsh"↙
/usr/bin/tcsh
/usr/share/doc/tcsh
/usr/doc/tcsh
/usr/lib/menu/tcsh
```

上面的搜索都没有区分文件与目录。如果要区分输出结果中的文件与目录，可以配合使用另一个常用的选项“ -type ”。

选项“ -type ” 指出需要查找的目录或文件名的类型必须是选项后所描述的类型。f 表示普通文件，d 表示目录，c 表示字符特殊文件，b 表示块特殊文件，l 表示符号链接文件。

例如：

```
ray@geecy:~$ find /bin -name "*sh" -type f↙
/bin/bash
/bin/tcsh
ray@geecy:~$ find /bin -name "*sh" –type l↙
/bin/rbash
/bin/sh
/bin/csh
```

虽然可以尝试分别使用 f 和 l，但还是不够清晰。配合选项“ –exec ”可以在搜索过程中

对搜索结果进行处理。例如：

```
ray@geecy:~$ find /bin -name "*sh" -exec ls -l {} \;↙
-rwxr-xr-x    1 root     root       511400 Apr  9  2002 /bin/bash
lrwxrwxrwx    1 root     root            4 Jan 30  2005 /bin/rbash -> bash
lrwxrwxrwx    1 root     root            4 Jan 30  2005 /bin/sh -> bash
-rwxr-xr-x    1 root     root       269768 Apr  2  2002 /bin/tcsh
lrwxrwxrwx   1 root    root   21 Jan 29  2005 /bin/csh -> /etc/alternatives/csh
```

选项“–exec”把搜索结果中的文件名替换命令“ls -l {}”中的“{}”，然后逐一执行该 ls 命令。“\;”的作用是转义成“;”。本例中实际执行的 ls 命令是：

```
ray@geecy:~$ ls -l /bin/bash; ls -l /bin/rbash; ls -l /bin/sh; ls -l /bin/tcsh↙
```

Bash Shell 支持在一行中输入多个命令，命令之间用分号隔开，按回车后一次执行这些命令。

【解决方案】

要快速浏览目录“/etc”下的所有子目录的结构和分布，Ray 可以使用带选项“-type”的 find 命令实现：

```
ray@geecy:~$ find /etc -type d↙
```

此方法虽然没有类似 tree 命令的树形结构的直观性，但具有通用性，因为一般在大多数的 Linux 发行版本都安装有 find 命令。

【问题的提出】

Linux 系统的/etc 目录保存了整个系统的配置文件。Ray 需要经常对该目录的所有内容进行备份。Ray 希望对备份的文件进行方便的管理。

【问题分析】

备份的文件有越多，其副作用越明显：一是严重浪费硬盘空间，因为通常情况下，就算是一个字节大小的文件也占用了 4KB 的空间；二是对于这些文件的搜索和再次复制的操作将大大降低运行速度。

比较好的解决方法是使用压缩功能，把整个备份的文件夹压缩为一个单独的文件，以方便管理。

*3.3.3 文件的压缩

■ 压缩

命令 tar 可以把整个目录的内容归并为一个单一的文件，但没有压缩。例如：

```
ray@geecy:~$ tar -cf bin.tar /bin↙
```

作用是把整个目录/bin 的内容归并为一个单一的文件，并指定该文件名是 bin.tar。使用后缀名“ .tar ”是通用的习惯。

选项“ -c ”表示归并操作。选项“ -f ”表示只对普通文件操作，而不包括设备文件等特殊文件。

命令 gzip 可以实现单个文件的压缩：

```
ray@geecy:~$ ls -l↙
total 2324
-rw-r--r--    1 ray      ray       2375680 Jan 26 02:25 bin.tar
ray@geecy:~$ gzip bin.tar↙
ray@geecy:~$ ls -l↙
total 996
-rw-r--r--    1 ray      ray       1015725 Jan 26 02:25 bin.tar.gz
```

gzip 的使用比较简单，默认是对给定的文件进行压缩。压缩完毕后，gzip 命令自动删除该文件。压缩后的文件名在原来的基础上增加后缀“ .gz ”。

从执行结果看出，bin.tar 压缩前的大小是 2375680 字节，压缩后是 1015725 字节。

■ 解压缩

按照常规，解压缩应该按照压缩的相反步骤进行，即先使用 gzip 解压，再使用 tar 从归并文件中抽取文件。例如：

```
ray@geecy:~$ ls -l↙
total 996
-rw-r--r--    1 ray      ray       1015725 Jan 26 02:25 bin.tar.gz
```

```
ray@geecy:~$ gzip -d bin.tar.gz↙
ray@geecy:~$ ls -l↙
total 2324
-rw-r--r--    1 ray      ray       2375680 Jan 26 02:25 bin.tar
```

选项“-d”用于指出是解压缩操作，而不是默认的压缩操作。

然后使用 tar 命令从归并文件中抽取文件。例如：

```
ray@geecy:~$ tar -xf bin.tar↙
ray@geecy:~$ ls -l↙
total 2328
drwxr-xr-x    2 ray      ray          4096 Jan 29  2005 bin
-rw-r--r--    1 ray      ray       2375680 Jan 26 02:25 bin.tar
```

选项“-x”用于指出是抽取操作。

tar 命令还提供一个更方便的选项“-z”，能够在抽取操作之前自动调用 gzip 先进行解压缩。例如：

```
ray@geecy:~$ ls -l↙
total 996
-rw-r--r--    1 ray      ray       1015725 Jan 26 02:25 bin.tar.gz
ray@geecy:~$ tar -zxf bin.tar.gz↙
ray@geecy:~$ ls -l↙
total 1000
drwxr-xr-x    2 ray      ray          4096 Jan 29  2005 bin
-rw-r--r--    1 ray      ray       1015725 Jan 26 02:25 bin.tar.gz
```

【解决方案】

由于/etc 目录下有些文件需要管理员权限才能读取（例如/etc/shadow），因此 Ray 应该暂时使用 su 命令切换到管理员帐号：

```
ray@geecy:~$ su -↙
```

然后：

```
geecy:~# tar -cf etc.tar /etc↙
geecy:~# ls -l↙
total 1964
-rw-r--r--    1 root     root      2007040 Jan 26 02:59 etc.tar
geecy:~# gzip etc.tar↙
geecy:~# ls -l↙
total 332
-rw-r--r--    1 root     root       333990 Jan 26 02:59 etc.tar.gz
```

事实上，tar 命令的选项“-z”不但可以应用在解压缩操作，还可以应用在压缩操作。

例如，Ray 可以使用一个 tar 命令即可以完成整个压缩过程：

```
geecy:~# tar -zcf etc2.tar.gz /etc↙
geecy:~# ls -l↙
total 332
-rw-r--r--    1 root     root       333982 Jan 26 03:08 etc2.tar.gz
```

按照习惯，一般指定的新文件名带有后缀“ .tar.gz ”。

3.4 练 习

1、 Tenny 需要以长格式使用三种排序方式显示目录/bin 的内容：
（1）以文件大小由大到小的次序排序；
（2）以文件大小由小到大的次序排序；
（3）以文件的最后修改时间的先后次序排序。

2、 Tenny 使用命令“ ls /tmp ”得到输出结果“ /tmp/abcd ”。请问这是一个文件还是一个目录？请使用实际的例子来解释。

3、 Tenny 通过一些参考资料了解到，ls 命令可以使用不同的颜色来表示输出结果中的不同文件类型。但在默认的情况下，他却没有看到过这些有不同颜色的结果。请问需要哪个选项指定输出结果使用不同的颜色来表示不同文件类型？
【提示】 使用 man 命令获取帮助。

4、 Tenny 在安装 Linux 系统的过程中按照系统的提示创建了一个普通用户帐号。但是后来他很少使用这个用户登录，连用户名是什么都忘记了。请只使用 cat 命令把该用户名找回来。

5、 Tenny 希望学习修改目录/etc 下的配置文件。为避免误操作导致系统崩溃，他需要先把目录/etc 整个复制到其个人目录下，并修改目录名为“ etc_bak ”。

6、 Tenny 已经成功使用命令“ mount /cdrom ”加载使用光盘。参数“ /cdrom ”只是指定了把光盘加载到的目的位置，如果计算机安装有不止一个的 CD-ROM，那么 Linux 系统应该能够选择其中一个使用。Tenny 查看了一些参考资料，了解到 Linux 系统中有个配置文件“ fstab ”存放有这些相关系信息。请只使用一个命令把这个文件找出来并显示其内容到屏幕上。

*7、由于目录 etc 下文本配置文件一般都是比较短小，因此使用普通复制的方式备份整个目录的内容将会导致严重浪费硬盘空间。Tenny 如何可以避免这种浪费？请给出解决方案，并具体实施。

第 4 章　Linux 权限管理

本章提要

- 用户管理
- 文件权限管理

【问题的提出】

Geecy 软件开发公司即将开始在 Linux 系统上进行两个项目的开发。开发人员 David 和 Peter 组成一个小组，负责 A 项目的开发；开发人员 Jack 和 Mike 组成另一个小组，负责 B 项目的开发。

系统管理员 Ray 需要为这四名开发人员分别建立用户帐号，并满足以下要求：

(1) 建立目录“/project_a”，该目录里面的文件只能由 David 和 Peter 两人读取、增加、删除、修改以及执行，其他用户不能对该目录进行任何的访问操作；

(2) 建立目录“/project_b”，该目录里面的文件只能由 Jack 和 Mike 两人读取、增加、删除、修改以及执行，其他用户不能对该目录进行任何的访问操作；

(3) 建立目录“/project”，该目录里面的文件只能由 David、Peter、Jack 和 Mike 四人读取、增加、删除、修改以及执行，其他用户只可以对该目录进行只读的访问操作。

【问题分析】

Linux 系统采用传统的 Unix 权限管理机制，对文件或目录的权限设置不能精确到每个用户。也就是，不能直接指定某个文件的权限类似于“对 X 和 Y 用户只读、对 M 和 N 用户可读可执行、对 P 和 Q 用户可读可写可执行”这么详细。

在 Linux 中文件权限的设置对象只有三种：文件的拥有者（1 个用户）、组的拥有者（若干个用户）以及其他用户（若干个用户）。但一个用户可以加入到多个用户组，从而可以形成比较复杂的关系，以满足现实需求。

因此，可以建立三个用户组：prj_a(组员有 David 和 Peter)、prj_b(组员有 Jack 和 Mike)和 prj(David、Peter、Jack 和 Mike)。然后：

(1) 对目录“/project_a”设置权限，允许组 prj_a 读取、增加、删除、修改以及执行，不允许其他用户进行任何的访问操作；

(2) 对目录“/project_b”设置权限，允许组 prj_b 读取、增加、删除、修改以及执行，不允许其他用户进行任何的访问操作；

(3) 对目录“/project”设置权限，允许组 prj 读取、增加、删除、修改以及执行，允许其他用户对该目录进行只读的访问操作。

4.1 用户管理

4.1.1 用户及权限

文本文件“/etc/passwd”保存有用户名、个人目录等资料。例如：

```
ray@geecy:~$ cat /etc/passwd↙
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
(中间省略一些行的内容)
ray:x:1000:1000:Debian User,,,:/home/ray:/bin/bash
identd:x:100:65534::/var/run/identd:/bin/false
sshd:x:101:65534::/var/run/sshd:/bin/false
tom:x:1001:100::/home/tom:/bin/bash
```

文件“/etc/passwd”以行为单位保存用户的信息。每行以冒号分隔为 7 个数据域。每个数据域的含义是：

a) 用户名。用户登录系统需要使用。
b) 密码。但由于在安装系统时候使用了影子(Shadow)密码的策略（请参考“1.2.4 进入安装过程”），因此此域都只显示一个特定的字符“x”。真正的密码被加密后存放在影子文件“/etc/shadow”。
c) 用户的权限 ID。具有权限 ID 值为 0 的用户称为系统管理员，默认配置下 root 用户的权限 ID 为 0。权限 ID 值非 0 的用户是普通用户。
d) 用户组的 ID。每个用户至少属于一个默认的分组。用户组的信息存放在文件“/etc/group”。
e) 注释。描述该用户的注释。例如可以是家庭地址、电话号码等信息。
f) 用户的个人目录。用户登录成功后系统读取该域，为用户设置当前工作目录为该个人目录。
g) 用户的登录 Shell。用户登录成功后系统读取该域，为用户启动一个 Shell。

一般情况下，管理员不必手工修改该文件。系统提供了命令“useradd”、“userdel”和“usermod”分别用于增加用户、删除用户和修改用户，还提供命令“passwd”用于修改用户密码。

命令 useradd 提供丰富的选项供管理员灵活增加用户：

```
geecy:~# useradd -m -u 2046 -g 1000 -d /tmp/hdf -s /bin/tcsh -c friend hdf↙
```

选项“-u”的功能是指定新增用户的 ID，选项“-g”的功能是指定新增用户所属于的默认组的 ID，选项“-d”的功能是指定新增用户的个人目录，选项“-s”的功能是指定新增用户的登录 Shell，选项“-c”的功能是指定新增用户的注释，选项“-m”的功能是指定为新增用户在文件系统上自动建立个人目录。

因此，上述命令增加用户 hdf，指定用户的 ID 是 2046，所属于的默认组的 ID 是 1000（与

ray 属于同一个组），个人目录是“ /tmp/hdf ”，登录 Shell 是“ /bin/tcsh ”。可以查看文件“ /etc/passwd ”以确认增加成功：

```
geecy:~# cat /etc/passwd↙
(省略前面内容)
ray:x:1000:1000:Debian User,,,:/home/ray:/bin/bash
identd:x:100:65534::/var/run/identd:/bin/false
sshd:x:101:65534::/var/run/sshd:/bin/false
tom:x:1001:100::/home/tom:/bin/bash
hdf:x:2046:1000:friend:/tmp/hdf:/bin/tcsh
```

此时，用户 hdf 还不能登录。接着查看文件“ /etc/shadow ”：

```
geecy:~# cat /etc/shadow↙
root:gmYZ7YQZK16gI:12812:0:99999:7:::
daemon:*:12812:0:99999:7:::
bin:*:12812:0:99999:7:::
(省略前面内容)
ray:gps7MNQruVPU2:12814:0:99999:7:::
identd:!:12812:0:99999:7:::
sshd:!:12812:0:99999:7:::
tom:J2pvuJUEhg3Gc:12808:0:99999:7:::
hdf:!:12808:0:99999:7:::
```

该文件的第 2 个数据域存放加密后的密码。如果该域是“！”则相应的用户不能登录。因为根据所使用的加密算法，不存在一个加密后为“！”的密码，就算使用穷举法破解也无效。

最后，管理员应该使用 passwd 命令为新用户设置密码。

4.1.2 用户的分组

相对于用户信息，用户组的信息要少些：

```
ray@geecy:~$ cat /etc/group↙
root:x:0:
daemon:x:1:
bin:x:2:
(中间省略一些行的内容)
users:x:100:
nogroup:x:65534:
ray:x:1000:
```

文件“ /etc/group ”以行为单位保存用户组的信息。每行以冒号分隔为 4 个数据域。每个数据域的含义是：

（1） 用户组的名称。

（2） 密码。用户组一般不需要设置密码。

(3) 用户组的 ID。

(4) 组的其他成员用户。因为每个用户至少属于一个默认的分组，也就是说，一个用户可以加入到多个组。该数据域以逗号为分隔保存组的成员用户（如果有的话）。默认情况下该数据域为空。

一般情况下，管理员不必手工修改该文件。系统提供了命令“groupadd”、“groupdel”和“groupmod”分别用于增加组、删除组和修改组。这些命令只能修改组的本身的基本信息，不能增加或减少组中所包含的成员用户。

例如：

```
geecy:~# groupadd  -g 10000 guest↙
geecy:~# cat /etc/group↙
（省略前面的内容）
users:x:100:
nogroup:x:65534:
ray:x:1000:
guest:x:10000:
```

该命令增加一个 ID 为 10000、名字为 guest 的组。

要修改某个用户所属于的组，可以使用 usermod 命令。例如：

```
geecy:~# usermod -G 0 hdf↙
geecy:~# usermod -G 0 tom↙
geecy:~# cat /etc/group↙
root:x:0:hdf,tom
daemon:x:1:
bin:x:2:
(中间省略一些行的内容)
users:x:100:
nogroup:x:65534:
ray:x:1000:
```

先修改用户 hdf，加入到组名为 root 的组（hdf 所属于的默认组不受影响）；然后修改用户 tom，加入到组名为 root 的组（tom 所属于的默认组不受影响）。

经过这样修改后，用户 hdf 的默认组是组 ID 为 1000 的用户组 ray，并同时属于组 ID 为 0 的用户组 root；用户 tom 的默认组是组 ID 为 100 的用户组 user，并同时属于组 ID 为 0 的用户组 root。也就是说，用户 root、hdf 和 tom 属于同一个用户组(root)。

4.2 文件权限管理

4.2.1 文件权限的描述

在 Linux 系统中，每一个文件都设置有相应的文件访问权限(FAP，File Access Permission)。可以使用带选项“ -l ”的 ls 命名查看 FAP。例如：

```
ray@geecy:~$ ls -l /bin/date↙
-rwxr-xr-x    1 root     root        25820 Jul 26  2001 /bin/date
```

第 1 列的内容（“ -rwxr-xr-x ”）共有 10 个字符，其中左边第 1 个字符（“ - ”）指出文件的类型是普通文件，剩余的 9 个字符（“ rwxr-xr-x ”）详细描述了文件“ /bin/date ”的权限设置。第 3 列的内容“ root ”描述了文件“ /bin/date ”的拥有者，第 4 列的内容“ root ”描述了文件“ /bin/date ”的组拥有者，即该文件所属于的用户组。

这 9 个字符（“ rwxr-xr-x ”）的具体描述方式是：前三个字符（“ rwx ”）描述文件拥有者对该文件具有的权限，中间三个字符（“ r-x ”）描述组拥有者对该文件具有的权限，最后三个字符（“ r-x ”）描述其他用户（既不是文件拥有者也不是组拥有者的其他用户）对该文件具有的权限。这些描述字符的具体含义在表 4-1 中列出。

表 4-1

符号	意义	对于文件的权限	对于目录的权限
r	读(Read)	显示文件内容	列出目录中的文件
w	写(Write)	编辑或删除文件	创建或删除文件和子目录
x	执行(Execute)	执行 Shell 脚本或二进制程序文件	搜索或进入目录

如果没有相应的读、写或执行的权限，则该位置使用字符“ - ”表示。因此文件“ /bin/date ”的 FAP 设置是：对于拥有者(用户 root)有读、写和执行的权限；对于组拥有者(组 root) 有读和执行的权限；对于其他用户有读和执行的权限。

另一个例子：

```
ray@geecy:~$ ls -l /etc/shadow↙
-rw-r-----    1 root     shadow        700 Jan 26 04:30 /etc/shadow
```

对于文件“ /etc/shadow ”，其拥有者 root 用户具有读和写（rw-）的权限，而属于组 shadow 的用户成员只有读（r--）的权限，除此之外的其他用户没有任何的操作权限（---）。

4.2.2 修改文件拥有者

只有系统管理员用户才能修改文件的拥有者和组拥有者。命令 chown 用于修改文件的拥有者。例如：

```
geecy:~# cp /kernelversion /tmp↙
```

```
geecy:~# ls -l /tmp/kernelversion↙
-rwxr-xr-x    1 root     root          451 Jan 26 06:22 /tmp/kernelversion
```

为了不影响系统，首先复制一个文件 kernelversion 到目录/tmp 下作为测试例子。该测试文件的拥有者是用户 root，组拥有者是组 root。根据该文件的 FAP 设置，如果普通用户 ray 对该文件执行删除命令：

```
ray@geecy:~# rm /tmp/kernelversion↙
```

那么只会得到出错提示信息，并不能删除该文件。因为对于该文件，ray 是属于“其他用户”，根据该文件的 FAP 设置，ray 只有读和执行的权限。例如：

```
ray@geecy:~# /tmp/kernelversion↙
2.2
```

然后，管理员使用 chown 命令把该文件的拥有者修改为用户 ray：

```
geecy:~# chown ray /tmp/kernelversion↙
-rwxr-xr-x    1 ray      root          451 Jan 26 06:22 /tmp/kernelversion
```

命令 chown 的第 1 个参数是用户名，第 2 个参数是需要修改拥有者的文件。经过这样的修改后，ray 成为该文件的拥有者，就可以全权控制该文件，当然也包括把它删除。

命令 chgrp 用于修改文件的组拥有者。例如：

```
geecy:~# ls -l /tmp/kernelversion↙
-rwxr-xr-x    1 ray      root          451 Jan 26 06:22 /tmp/kernelversion
geecy:~# chgrp users /tmp/kernelversion↙
geecy:~# ls -l /tmp/kernelversion↙
-rwxr-xr-x    1 ray      users         451 Jan 26 06:22 /tmp/kernelversion
```

命令 chgrp 把文件“/tmp/kernelversion”的组拥有者修改为组 users。

命令 chown 和 chgrp 对目录的拥有者和组拥有者的修改类似于对文件的修改。同样，这两个命令也提供一个选项“-R”用于以递归方式实现对整个目录的所有内容（所有子目录和文件）全部修改。请参考 cp 命令的使用。

4.2.3 修改文件访问权限

用户 tom 属于用户组 users。文件‘/tmp/kernelversion’目前的权限设置是属于组 users 的用户和其他的用户具有同样的权限，即都只有读和执行的权限（r-x）。

可以继续修改该文件的 FAP，使得 tom 不是该文件的拥有者也能够像 ray 一样对该文件具有修改和删除的权限，即需要把该文件的组拥有者的权限在原来的“r-x”基础上增加“w”权限。

修改文件访问权限可以使用 chmod 命令。

系统管理员可以修改整个文件系统的所有文件的 FAP，普通用户只能修改拥有者是自己的那些文件。因此，用户 ray 就可以运行下面的命令：

```
ray@geecy:~# ls -l /tmp/kernelversion↙
```

```
-rwxr-xr-x    1 ray      users         451 Jan 26 06:22 /tmp/kernelversion
ray@geecy:~# chmod g+w /tmp/kernelversion↙
ray@geecy:~# ls -l /tmp/kernelversion↙
-rwxrwxr-x    1 ray      users         451 Jan 26 06:22 /tmp/kernelversion
```

执行 chmod 命令后，该文件的 FAP 由原来的“ rwxr-xr-x ”变为“ rwxrwxr-x ”。

chmod 命令的第 1 个参数指出如何修改访问权限，第 2 个参数指出需要修改访问权限的文件。“ g+w ”表示在该文件的组拥有者权限的基础上增加一个写的权限。

这是一种通过符号描述修改 FAP 的方式。

在这种方式中，u、g 和 o 分别表示文件的拥有者(User)、组拥有者(Group)和其他用户(Others)；符号“ + ”和“ - ”分别表示增加权限和减去权限。

例如：

```
ray@geecy:~# ls -l /tmp/kernelversion↙
-rwxrwxr-x    1 ray      users         451 Jan 26 06:22 /tmp/kernelversion
ray@geecy:~# chmod o-rx /tmp/kernelversion↙
ray@geecy:~# ls -l /tmp/kernelversion↙
-rwxrwx---    1 ray      users         451 Jan 26 06:22 /tmp/kernelversion
```

chmod 命令使用“ o-rx ”作参数把文件“ /tmp/kernelversion ”的 FAP 由“ rwxrwxr-x ”修改为“ rwxrwx--- ”。现在除用户 ray 和组 user 的成员之外的其他用户对该文件没有任何的访问权限。

也可以组合使用“ u ”、“ g ”和“ o ”。例如：

```
ray@geecy:~# ls -l /tmp/kernelversion↙
-rwxrwx---    1 ray      users         451 Jan 26 06:22 /tmp/kernelversion
ray@geecy:~# chmod u-x,g-w,o+x  /tmp/kernelversion↙
ray@geecy:~# ls -l /tmp/kernelversion↙
-rw-r-x--x    1 ray      users         451 Jan 26 06:22 /tmp/kernelversion
```

组合使用需要使用逗号隔开，例如“ u-x,g-w,o+x ”。

使用这种符号描述的方式修改 FAP 的好处是用户不需要了解原有的文件权限，用户只需要专注于所修改的局部权限。这种方式关注的是修改过程，即如何增加某个权限、如何减少某个权限等。

另一种方式称为绝对方式。这种绝对方式关心的是该文件的最终 FAP 是如何样子。至于由原有的 FAP 变化到新的 FAP 这个过程是如何通过增加或减少权限实现的，用户不需要使用参数指出。

这种绝对方式是通过二进制描述实现的：

（1）r、w 和 x 表示二进制位中的 1，“ - ”表示二进制位中的 0。

（2）每三个二进制位转换成对应的十进制值。

例如：

```
ray@geecy:~# ls -l /tmp/kernelversion↙
```

```
-rw-r-x--x    1 ray      users         451 Jan 26 06:22 /tmp/kernelversion
```

文件“/tmp/kernelversion”的 FAP 是“rw-r-x--x”。把该 FAP 每三个分为三组“rw-”、“r-x”和“--x”，对应的二进制描述分别是“110”、“101”和“001”。再转换为十进制，分别是“6”、“5”和“1”。因此该文件 FAP 也可以记作“651”。

现在要求把该文件的 FAP 修改为“rwxr-x---”，也就是“750”。使用下面命令：

```
ray@geecy:~# chmod 750 /tmp/kernelversion↙
ray@geecy:~# ls -l /tmp/kernelversion↙
-rwxr-x---    1 ray      users         451 Jan 26 06:22 /tmp/kernelversion
```

使用绝对方式修改 FAP，其修改结果不受原来的 FAP 的影响。

命令 chmod 对目录的访问权限的修改类似于对文件的修改。同样，该命令也提供一个选项“-R”用于以递归方式实现对整个目录的所有内容（所有子目录和文件）全部修改。请参考 cp 命令的使用。

【解决方案】

按照前面的问题分析，先增加用户组：

```
geecy:~# groupadd -g 2000 prj↙
geecy:~# groupadd -g 2001 prj_a↙
geecy:~# groupadd -g 2002 prj_b↙
```

再增加用户：

```
geecy:~# useradd -m -u 3001 -g 2001 -G 2000 david↙
geecy:~# useradd -m -u 3002 -g 2001 -G 2000 peter↙
geecy:~# useradd -m -u 3003 -g 2002 -G 2000 jack↙
geecy:~# useradd -m -u 3004 -g 2002 -G 2000 mike↙
```

修改用户密码：

```
geecy:~# passwd david↙
geecy:~# passwd peter↙
geecy:~# passwd jack↙
geecy:~# passwd mike↙
```

建立目录：

```
geecy:~# mkdir /project↙
geecy:~# mkdir /project_a↙
geecy:~# mkdir /project_b↙
```

修改目录的组拥有者：

```
geecy:~# chgrp prj /project↙
geecy:~# chgrp prj_a /project_a↙
geecy:~# chgrp prj_b /project_b↙
```

修改目录的 FAP：

```
geecy:~# chmod 574 /project↙
geecy:~# chmod 570 /project_a↙
geecy:~# chmod 570 /project_b↙
```

4.3 练 习

1、 复制文件“ /etc/hosts ”到目录“ /tmp ”，然后设置其读写权限为：自己可以读、写和执行，组拥有者的用户可以读和执行，其他用户只可以执行。

2、 复制文件“ /etc/hosts ”到目录“ /tmp ”，然后只使用 chown 命令修改其拥有者为用户 ray，组拥有者为组 users。要求不得使用 chgrp 命令。

【提示】 使用 man 命令获取帮助。

3、 Tenny 一个人使用 Linux 系统，他既是系统管理员，又是普通用户。为了系统的稳定使用，他需要使用管理员用户为自己创建的两个用户帐号 tenny 和 ten。Tenny 平时使用这两个用户登录使用系统，为了这两个用户交换和共享使用的方便，还需要达到如下的要求：

(1) 在系统中建立一个目录“ /mydir ”；

(2) 设置目录“ /mydir ”的权限为：该目录里面的文件只能由 tenny 和 ten 两个用户读取、增加、删除、修改以及执行，其他用户不能对该目录进行任何的访问操作。

第 5 章　在 Linux 下开发应用程序

本章提要

- vi 的基本使用
- vi 的文本搜索
- 使用编译器

5.1　使用文本编辑器

【问题的提出】

Geecy 软件开发公司的开发人员 Tom 将要参与一个新项目的开发。现在他需要在 Linux 平台下为自己选择一个适合编写源代码和撰写文档的全屏幕文本编辑器。要求功能强大、简单易用以及容易入门。

【问题分析】

开发人员需要编写源代码和撰写文档，而对于系统管理员来说，由于系统的配置是通过文本文件的描述进行设定，因此也经常使用文本编辑器。因而，无论是开发人员还是系统管理员都离不开文本编辑器。可以毫不夸张地说，大部分的用户使用 Linux 系统，大多数的时间是在使用文本编辑器。好的文本编辑器会令工作事半功倍。

目前在 Linux 下存在很多的文本编辑器，例如：cooledit、ee、elvis、emacs、jed、joe、vi、vim、xedit、xemacs 等等。

文本编辑器的基本操作一般包括：文本的输入、选定、复制、删除、粘贴以及保存退出等。一般来说，用户根据自己的习惯和爱好等因素，精心挑选一个适合的文本编辑器作为自己的常用编辑器即可，没有必要经常转换。

在 Linux 下，有两大著名的文本编辑器 vi 与 emacs，大部分的用户会选择其中之一作为常用的编辑器。

5.1.1　vi 简介

vi 是一个可视化(Visual)的全屏幕文本编辑器，默认安装在各种 UNIX 系统上。Linux 的各种发行版本都安装有 vi 的仿真或改进版本。

vi 的功能非常强大而操作简便有效，在各种 UNIX 系统和各种 Linux 发行版本中都可以使用 vi 对文本文件进行编辑，而且 vi 能兼容工作于各种的物理终端和仿真终端。基于这三个重要的理由，推荐用户（特别是初学者）选用并熟练掌握 vi。

5.1.2 vi 的基本使用

可以指定一个文件名作参数运行 vi：

```
tom@geecy:~$ vi hello.c↙
```

如果当前工作目录下存在一个名为 hello.c 的文件，则 vi 显示其内容在屏幕上。如果不存在这个文件，则显示如下所示的空屏幕：

```
-
~
~
~
~
~
"hello.c" [New File]
```

vi 有两种工作状态：编辑(Edit)状态与命令(Command)状态。每次运行 vi，它总是默认处于命令状态。

如果处于命令状态，那么用户不能输入任何的文本内容。任何的键盘输入都被看作是命令序列，不显示在编辑窗口。vi 没有任何可视化的功能菜单，所有的任务都是由一个或几个的字符序列组成的键盘命令来完成。

屏幕的最后一行是状态行（“hello.c”是当前编辑的文件名，“[New File]”表明该文件是一个新创建的文件，还没有经过保存操作）。

如果处于编辑状态，或者你根本不清楚当前在什么状态，任何时候都可以按“Esc”键切换到命令状态。连续按“Esc”键若干次，其效果等同于一次。为确保真正进入命令状态，建议用户连续按两到三次。

处于命令状态，可以用 i 命令进入编辑状态。即先按“Esc”键进入命令状态后再按小写的“i”键。此时屏幕的最后一行的状态栏由

```
"hello.c" [New File]
```

变为

```
-- INSERT --
```

这表明 vi 已经处于编辑状态：

```
-
~
~
~
-- INSERT --
```

处于编辑状态，用户就可以像在使用 Microsoft Windows 的记事本一样自由地输入文本的正文内容。输入完一行内容，按回车键就会开出新行供用户输入下一行的内容。作为例子，请输入如下的 5 行文本内容：

```
#include <stdio.h>
main()
```

```
{
    printf("%s\n", "Hello, world! ");
}
~
~
~
-- INSERT --
```

这是一个简单的 C 语言源程序（关于 C 语言程序的编译和运行请参考“5.2.1 ”）。

最后，用户按“Esc”键进入命令状态，使用:x 命令保存文件并退出：

```
#include <stdio.h>
main()
{
    printf("%s\n", "Hello, world! ");
}
~
~
~
:x
```

至此，用户仅仅用了 i 和:x 两个命令就实现了使用 vi 编写 C 语言源程序的任务。

【注意】 vi 的命令分为两类：以“:”（冒号）或“/” （斜杠）或“?”(问号）开头的命令将在此状态行显示，按回车键后发生作用（例如:x）；另一类命令则不在状态栏显示，也无需按回车键，直接发生作用（例如 i)。

用户也可以在命令提示符下使用 cat 命令（或 more、less）查看文件“hello.c”的内容:

```
tom@geecy:~$ cat hello.c↙
#include <stdio.h>
main()
{
    printf("%s\n", "Hello, world! ");
}
```

用户还可以再次使用 vi 打开文件“hello.c”，以进行增加、删除或修改等操作：

```
tom@geecy:~$ vi hello.c↙
```

由于文件“hello.c”已经存在，因此屏幕将显示文件“hello.c”的内容：

```
#include <stdio.h>
main()
{
    printf("%s\n", "Hello, world! ");
```

```
}
~
~
~
"hello.c" 5L, 65C
```

vi 同样是处于命令状态。状态行除显示文件名外还显示两个统计信息：该文件有 6 行，共 66 字节。

要想轻松简单的使用 vi 编辑文本文件，初学者需要把握以下的两个要点：

(1) 使用 i 命令进入编辑状态后，使用 vi 就像使用 Microsoft Windows 的记事本一样简单，只需结合方向键（上、下、左、右）、删除键(Del)、后退键(Backspace)以及回车键(Enter)等控制键进行文本的增加、删除和修改操作；

(2) 按“ Esc ”键返回命令状态，使用:x 命令保存文件并退出。

为检验对这两个要点是否真正把握，请把前面的 hello.c 的内容修改为如下所示：

```
#include <stdio.h>
int main()
{
    char message_str[]="Hello,world! I am using vi...";
    char format_str[]="%s\n";
    printf(format_str,message_str);
    return 0;
}
~
~
~
-- INSERT --
```

最后保存文件并退出 vi。

5.1.3 vi 的文本搜索

对于在一个很长的文件中定位出某个词或词组这类任务，总不能让用户一行一行或一页一页的凭眼睛去比较对照搜索吧。就算是要在一页之内搜索某个词组，有时也足以让用户“ 痛不欲生 ”。这显然不能发挥计算机的自动化作用。

vi 提供了强大的全文搜索功能供用户使用。只要用户给定一个待匹配的字符串（即搜索关键字），vi 将会从当前的光标位置向下（或向上）进行逐个匹配，如果匹配成功，那么光标将定位到该位置，等待用户的下一步处理。

搜索命令的语法比较简单，如表 5-1 所示。

表 5-1 搜索命令

命 令	功 能
/pattern	从光标位置向下查找包含模式 pattern 的下一行
?pattern	从光标位置向上查找包含模式 pattern 的上一行

■ 精确搜索

精确搜索所使用的关键字是一个具体的字符串。例如，使用 vi 重新打开文件“ hello.c ”，以作为搜索的例文：

```
tom@geecy:~$ vi hello.c↙
```

这时候 vi 处于默认的命令状态，屏幕显示有文件内容和状态行信息：

```
#include <stdio.h>
int main()
{
    char message_str[]="Hello,world! I am using vi...";
    char format_str[]="%s\n";
    printf(format_str,message_str);
    return 0;
}
~
"hello.c" 8L, 158C
```

需要注意，光标位置默认处于第一行第一个字符。

在命令状态下，用户可以发出搜索命令。例如要从当前光标位置向下搜索关键字为“ str ”的位置，则直接输入以“ / ”为首的搜索命令：

```
/str
```

状态行将显示该命令：

```
#include <stdio.h>
int main()
{
    char message_str[]="Hello,world! I am using vi...";
    char format_str[]="%s\n";
    printf(format_str,message_str);
    return 0;
}
~
/str
```

再按回车键，搜索命令将执行。如果搜索匹配成功，那么光标将定位在最先匹配成功的该内容的第一个字符。在本例子中，搜索匹配成功，光标定位在第四行上：

```
#include <stdio.h>
```

```
int main()
{
    char message_str[]="Hello,world! I am using vi...";
    char format_str[]="%s\n";
    printf(format_str,message_str);
    return 0;
}
~
~
/str
```

这时候，用户可以使用 i 命令进入编辑状态，对当前光标的内容进行编辑，或是移动光标到其他地方进行编辑。

编辑完成后，如果有需要，用户也可以继续搜索。同样，需要进入命令状态，即按“ Esc ”进入。

如果需要搜索的关键字不变，即仍然搜索字符串“ str ”，那么只需简单发出单独的命令“ / ”即可，vi 将从当前光标位置向下搜索同样的关键字“ str ”。

若匹配成功，则光标定位在最先匹配成功的内容的第一个字符：

```
#include <stdio.h>
int main()
{
    char message_str[]="Hello,world! I am using vi...";
    char format_str[]="%s\n";
    printf(format_str,message_str);
    return 0;
}
~
~
/
```

当然，这时候用户可以选择从当前光标位置向上搜索，如果搜索关键字不变，那么只需要把搜索命令由“ / ”换成“ ? ”即可。

■ 模糊搜索

另一种搜索称为模糊搜索，其搜索的关键字含有一些具有特殊含义的描述字符，称之为模式(Pattern)，也称为正则表达式(Regular Expression) ，或正规表达式。这些具有特殊含义的描述字符称为元字符。

精确搜索所用到的关键字是一个具体的字符串，就是一个特殊的模式，即只包含普通字符而没有元字符。

表 5-2 列出了正则表达式常用的元字符。

表 5-2 正则表达式的元字符

字符	含　义
.	匹配任何一个字符（回车换行符除外）
*	匹配零个或零个以上的与前一个字符相同的字符
^	匹配下一个字符必须出现在某行的行首
$	匹配前一个字符必须出现在某行的行末
[ ]	匹配方括号里面的若干个字符的其中一个字符
[-]	匹配方括号里面的某一范围字符的一个字符
[^]	匹配除了方括号里面的字符之外的其他任何一个字符
\	把后面的元字符转义，使其不再具有特定的含义，变成普通字符

再次使用 vi 打开文件“ hello.c ”作为搜索的例文：

```
#include <stdio.h>
int main()
{
    char message_str[]="Hello,world! I am using vi...";
    char format_str[]="%s\n";
    printf(format_str,message_str);
    return 0;
}
~
/ma.n
```

如果分别以表 5-3 所示的正则表达式的例子作为模式在文件“ hello.c ”中进行搜索，那么可以得到的匹配次数分别等于列于表 5-3 的解析一栏的方括号中的数字。

表 5-3 正则表达式的例子

例　子	解　析
in.	[4] 包含 in，后接任何一个字符的模式
ma.n	[1] 包含 ma，后接任何一个字符，再后接 n 的模式
^in	[1]　in 出现在某行的行首的模式
";$	[2]　";出现在某行的行末的模式
ma[it]	[3] 包含 ma，后接 i 或 t 的模式
in[b-p]	[1] 包含 in，后接从 b 到 p 中任一个字符的模式
in[^b-p]	[3] 包含 in，后接不是从 b 到 p 中任一个字符的模式
in[^tc]	[1] 包含 in，后接不是 t 也不是 c 的任一个字符的模式
str\[\]	[2] 包含 str[]的模式
[A-Za-z][A-Za-z]*	匹配任何的英文单词（以空格分隔作为一个单词）

【提示】 如果需要搜索的模式不变，那么只需简单发出单独的命令“/”或“?”即可。

*5.1.4 vi 使用进阶

虽然现在可以使用 vi 工作了，但是工作效率却不怎么样。例如，假设要删除连续 10 行的内容，用户可以怎么做？就算以平均每行 30 个字符计算，那么用户就需要按“Del”键 300 次了，哪来的高效率呢？

事实上，类似这样的编辑任务在 vi 中可以轻松高效完成。vi 提供了一些功能强大的但容易记忆的命令供用户使用。

■ 编辑命令

表 5-4 列出了五个常用的编辑命令。利用这些命令的组合，大部分的复杂编辑任务都可以在瞬间快速高效完成。

表 5-4 编辑命令

命 令	功 能
[N]x	(Expurgate)删除从光标位置开始的连续 N 个字符（并复制到编辑缓冲区）
[N]dd	(Delete)删除从光标位置开始的连续 N 行（并复制到编辑缓冲区）
[N]yy	(Yank)复制从光标位置开始的连续 N 行到编辑缓冲区
p	(Put)从编辑缓冲区复制文本到当前光标位置（即粘贴）
u	(Undo)取消上一次操作（即恢复功能）

这些命令都是在命令状态下使用（下同）。编辑命令操作的对象是当前光标位置。如果 N 等于 1,那么 N 可以省略。

例如，如果要删除从光标位置开始的连续 15 个字符，则具体的命令为 15x，即连续按下三个键“1”、“5”和“x”；但如果只是删除光标所在位置的一个字符，则 N 可以省略，具体的命令为 x，即只需要按下键“x”。

同样，如果要删除从光标位置开始的连续 13 行内容，可以发出命令 13dd，即连续按下四个键“1”、“3”、“d”和“d”。如果只是删除光标位置所在的一行，则 N 可以省略，具体的命令为 dd，即只需要连续按下两个键“d”和“d”。

复制操作类似于删除操作，只是命令由“dd”改为“yy”。两种操作的共同点都是把需要操作的若干行内容暂时保存在编辑缓冲区（一块内存区域）；不同点是，复制操作在执行完毕后仍保留原来若干行内容，而删除操作在操作完毕后把原来若干行内容删除。

当复制或删除某些内容后，可以移动光标到需要的某个地方，然后使用 p 命令把编辑缓冲区的内容提取出放到光标的当前位置。

命令“x”、“dd”、“yy”和“p”所影响的操作都可以使用命令“u”来取消，即具有恢复功能。

【注意】 编辑缓冲区只有一个，每一次复制某些内容到编辑缓冲区后都会把编辑缓冲区旧的内容覆盖。

■ 光标命令

表 5-5 列出了一些常用的与光标操作相关的命令。

基于两个主要的理由，建议用户使用“h”、“j”、“k”和“l”四个命令代替使用键盘上的原有的四个方向键：

一是，如果需要高速高效的编辑文本，那么十个手指不应该为了上下左右移动光标而频繁离开键盘上的 26 个字母键去按方向键；

二是，对于大多数通过网络使用仿真终端软件远程登录 Linux 使用 vi 的情况，有些仿真终端特别是物理终端对键盘上原有的四个方向键的支持不是很好（即兼容性问题），要想使用 vi 具有通用性，建议使用“h”、“j”、“k”和“l” 四个命令。

表 5-5 光标命令

命 令	功 能
h	方向键，向左移动光标一个字符的位置，相当于键“←”
j	方向键，向下移动光标到下一行的位置，相当于键“↓”
k	方向键，向上移动光标到上一行的位置，相当于键“↑”
l	方向键，向右移动光标一个字符的位置，相当于键“→”
:N	移动光标到第 N 行（N 待定）
1G	移动光标到文件的第 1 行
G	移动光标到文件的最后 1 行
:set number	设置显示行号
:set nonumber	取消显示行号

关于光标的快速移动，有三个命令很好用。G 命令直接可以令光标直接定位到文件的最后一行，而 1G 命令（连续按下两个键“1”和“G”）可以令光标直接定位到文件的第 1 行。另外，只要你知道目标是在第几行，也可以使用:N 命令令光标直接定位到文件指定行。例如，命令:35（连续按下三个键“:”、“3”和“5”）可以令光标直接定位到文件第 35 行。

大写字母 G 的按键建议使用<Shift> + g 实现，而不建议使用<Caps Lock>。

对于开发人员来说，程序调试经常需要了解行号。vi 提供了显示行号的功能（只是显示，并不是文件内容的一部分，保存文件的操作也不包括行号）。命令:set number 用户设置显示行号：

```
1 #include <stdio.h>
2 int main()
3 {
4    char message_str[]="Hello,world! I am using vi...";
5    char format_str[]="%s\n";
6    printf(format_str,message_str);
```

```
    7    return 0;
    8 }
~
:set number
```

使用命令:set nonumber 可以取消行号的显示。

■ 文件命令

表 5-6 列出了一些常用的与文件操作相关的命令。

表 5-6 文件命令

命 令	功 能
:q	(Quit)退出没有被修改的文件（若文件被修改了而没有保存，则此命令无效）
:q!	强制退出，且不保存修改过的部分
:w	(Write)保存文件，但不退出
:w!	强制保存文件，但不退出
:x	(Exit)保存文件并退出
:x!	强制保存文件并退出
:w File	另存为 File 给出的文件名，不退出
:w! File	强制另存为 File 给出的文件名，不退出
:r File	(Read)读入 File 指定的文件内容插入到光标位置

使用频率比较高的文件命令为：“ :q! ”、“ :w ”、“ :x ”和“ :w File ”。

用户打开一个文件并进行了一些修改，然后不希望保存这些修改而退出 vi，那么就使用“ :q! ”命令。

用户打开一个文件并进行了一些修改，然后希望保存这些修改以防止意外丢失并且需要继续编辑，那么就使用“ :w ”命令。经常使用该命令是一个好习惯。

用户打开一个文件并进行了一些修改，然后希望保存这些修改并结束工作退出 vi，那么就使用“ :x ”命令。

用户打开一个文件并进行了一些修改，然后希望保存这些修改以防止意外丢失并且不希望影响原有的文件（或者对原有的文件没有修改权限），那么就使用“ :w File ”命令。需要使用具体的文件路径代替 File（表 6-4 的另两个类似命令也一样），相对路径和绝对路径都可以。例如“ :w ~/hello.cpp ”或者“ :w /tmp/test.c ”。需要注意，以后的编辑操作都是针对新命名的文件。

一般来说，如果某个正在编辑的文件其文件拥有者就是正在对这个文件进行编辑的用户，但是文件的权限设置对该用户没有修改权限，那么保存文件的时候应该选用带有叹号“ ! ”的那命令进行强制保存。例如对于文件“ /etc/shadow ”，Linux 系统为防止该文件被意外的误操作修改，其默认的权限设置是“ r-------- ”：

```
tom@geecy:~$ ls -l /etc/shadow↙
```

```
-r--------    1 root     root          798 Jan 18 22:33 /etc/shadow
```

即如果系统管理员（root 用户）对该文件进行修改，那么保存的时候应该选用命令“ :w! ”或“ :x! ”才能成功。

需要注意，普通用户能对拥有者是自己的那些文件进行强制保存，而系统管理员用户可以对任何的文件进行强制保存。

■ 状态命令

表 5-7 列出了一些常用的与状态转换操作相关的命令。其中最常用的状态命令是“ i ”、“ Esc ”和“ :! Command ”。

表 5-7 状态命令

命 令	功 能
a	(Append)进入编辑状态，从当前光标之后的位置开始插入键盘输入的字符
i	(Insert)进入编辑状态，从当前光标之后的位置开始插入键盘输入的字符
o	(Open)进入编辑状态，并插入一新行，光标移到该新行的行首，以后键盘输入的字符将插入到光标位置
ESC	进入命令状态
:! Command	在 vi 中执行外部命令 Command，按回车键可以返回 vi 继续工作

命令“ :! Command ”的功能非常强大且使用方便。

当用户正在使用 vi 编辑文件，希望在不退出 vi 的条件下临时执行某些外部命令并观看执行结果，然后接着编辑工作，这时候使用该命令就非常适合。需要使用具体的命令代替 Command。例如“ :! ls /bin ”和“ :! pwd ”等。一般来说，能够在命令提示符下执行的命令都可以在这里使用。

开发人员经常需要在编辑源代码的时候执行一些外部命令对该源代码进行编译和运行（请参考“ *5.2 使用编译器 ”），如果有出错信息就可以继续修改该源代码，而不用反复退出和进入 vi，从而大大提高工作效率。

【解决方案】

的确，vi 的功能非常强大而操作简便有效，对各种终端兼容性好，而且可伸缩性极强。一般的小文件的操作，只需要用到“ 5.1.2 vi 的基本使用 ”和“ 5.1.3 vi 的文本搜索 ”中的精确搜索这两部分容易上手的内容就足以轻松完成任务。而当用户需要高效率工作，则可以使用“ *5.1.4 vi 使用进阶 ”和“ 5.1.3 vi 的文本搜索 ”中的模糊搜索这两部分功能强大的内容。

毫无疑问，Tom 立刻选定了 vi 作为自己的首选编辑器。

*5.2 使用编译器

【问题的提出】

Geecy 软件开发公司的开发人员 Tom 已经选定 vi 作为自己的常用文本编辑器，并使用 vi 亲自编写了一个简单的 C 语言源程序文件“hello.c”保存在其个人目录下。现在，他需要把该源程序转换成为可在 Linux 下运行的程序。

【问题分析】

C 语言源程序需要经过编译和链接这两个过程才能转换成二进制可执行程序。大多数的 C 编译器都可以把这两个步骤合二为一，直接生成可执行程序。一般在 Unix 系统中使用的 C 编译器是 cc（C Compiler 的缩写）。由于版权问题，在 GNU/Linux 中一般不包含有该编译器。在各个 Linux 发行版本中广泛使用的 C 编译器名为 gcc（GNU cc）。gcc 能够很好的兼容 cc 编译器。其中的区别只是 gcc 属于自由软件，而 cc 属于商业软件。Debian GNU/Linux 自带的 gcc 编译器的版本是 2.95。

5.2.1 使用 gcc

gcc 编译器的用法可以很简单，也可以很复杂。之所以复杂，是因为 gcc 提供很多功能强大的选项（诸如一些优化代码生成的选项、优化运行速度的选项等）供程序员使用。一般版本比较新的 gcc 已经集成了 C 编译器和 C++编译器，gcc 根据源程序的后缀名来决定使用哪一种语言的编译器进行编译工作。后缀名为“.c”(小写）的文件被 gcc 认为是 C 语言的源程序文件。

使用 gcc 的最简单方法是只使用一个后缀名为“.c”的文件作参数。例如：

```
tom@geecy:~$ ls -l↙
-rw-r--r--    1 tom      tom           158 Jan 26 09:49 hello.c
tom@geecy:~$ gcc hello.c↙
tom@geecy:~$ ls -l↙
-rwxr-xr-x    1 tom      tom          4871 Jan 26 10:42 a.out
-rw-r--r--    1 tom      tom           158 Jan 26 09:49 hello.c
```

gcc 编译出来的可执行程序默认是 a.out。接着运行该程序：

```
tom@geecy:~$ ./a.out↙
Hello,world! I am using vi...
```

用户可以在编译的时候修改默认的名字 a.out。例如：

```
tom@geecy:~$ gcc -o hello hello.c↙
tom@geecy:~$ ls -l↙
-rwxr-xr-x    1 tom      tom          4871 Jan 26 10:42 hello
-rw-r--r--    1 tom      tom           158 Jan 26 09:49 hello.c
```

选项“ -o ”的作用是指定一个名字作为生成的二进制可执行程序的名字。然后运行可执行文件 hello：

```
tom@geecy:~$ ./hello↙
Hello,world! I am using vi...
```

【注意】由于 Linux 系统默认配置的搜索路径 PATH 不包含当前工作目录，因此这里需要添加“ ./ ”指定确切的路径。当然，用户也可以使用绝对路径，例如“ /home/tom/hello ”。另外，gcc 生成的可执行文件的 FAP 已经默认设置有执行权限。如果某些版本的编译器不是这样，请再使用 chmod 命令增加执行权限。

5.2.2 使用 g++

下面是一个简单的 C++语言源程序：

```
tom@geecy:~$ cat hello.cc↙
#include <iostream>
#include <string>
int main()
{
        string message_str="Hello,world! I am using C++ Lanugage!";
        cout << message_str << endl;
        return 0;
}
```

g++是一个 C++版本的 gcc 编译器。g++要求 C++语言源程序文件带有后缀名“ .cc ”。编译 C++语言源程序与编译 C 语言源程序类似：

```
tom@geecy:~$ ls -l↙
-rw-r--r--    1 tom      tom           156 Jan 26 11:19 hello.cc
tom@geecy:~$ g++ hello.cc↙
tom@geecy:~$ ls -l↙
-rwxr-xr-x    1 tom      tom         23434 Jan 26 11:19 a.out
-rw-r--r--    1 tom      tom           156 Jan 26 11:19 hello.cc
```

g++编译出来的可执行程序默认也是 a.out。接着运行该程序：

```
tom@geecy:~$ ./a.out↙
Hello,world! I am using C++ Lanugage!
```

g++也提供选项“ -o ”。例如：

```
tom@geecy:~$ g++ -o hw hello.cc↙
tom@geecy:~$ ls -l↙
-rwxr-xr-x    1 tom      tom         23434 Jan 26 11:19 hw
-rw-r--r--    1 tom      tom           156 Jan 26 11:19 hello.cc
```

然后运行可执行文件 hw：

```
tom@geecy:~$ ./hw↙
Hello,world! I am using C++ Lanugage!
```

【注】如果 gcc 已经集成了 g++，那么系统一般会提供一个同样名字的 g++脚本供用户使用，该脚本负责调用 gcc 并传递一些编译 C++所需要的选项。不管集成与否，对用户来说是透明的。

5.3 练 习

（请按照顺序完成下列任务）

1、**使用** vi **打开一个新文档，并输入以下两行内容：**

> Linux is an operating system that was initially created as a hobby by a young student, Linus Torvalds, at the University of Helsinki in Finland. Linus had an interest in Minix, a small UNIX system, and decided to develop a system that exceeded the Minix standards.
>
> The kernel, at the heart of all Linux systems, is developed and released under the GNU General Public License and its source code is freely available to everyone. It is this kernel that forms the base around which a Linux operating system is developed.

2、**发出命令显示行号。**

3、**保存到文件** AboutLinux**，不退出。**

4、**删除一句“** It is this kernel that forms the base around which a Linux operating system is developed. **”。**

5、**查找单词“** Finland **”。**

6、**把第一行的“** Finland **”后的内容变成独立的一行。现在共有三行内容。**

7、**复制第二行的内容到文档的最后。**

8、**删除第三行的内容。**

9、**恢复被删除的一行内容。**

10、**查找所有的“** Minix **”并全部改为“** MINIX **”。**

11、**不保存并退出** vi**。**

12、**使用** vi **再次打开文件** AboutLinux**。应该显示原来的两行内容。**

13、**在第二行后插入一行“** He began his work in 1991 when he released version 0.02 and worked steadily until 1994 when version 1.0 of the Linux Kernel was released. **”。**

14、**在文档的最后增加一行‘** There are now literally hundreds of companies and organizations and an equal number of individuals that have released their own versions of operating systems based on the Linux kernel. **”。**

15、**保存并退出** vi**。**

16、**设置文件** AboutLinux **的权限为** 400 **（“** r-------- **”）。**

17、使用 vi 再次打开文件 AboutLinux，并在文档最后增加一行“ More information on the kernel can be found at our sister site, LinuxHQ and at the official Linux Kernel Archives. ”。并保存文档后退出。

第 6 章　Linux 组合命令

本章提要

- 标准文件简介
- 文件重定向
- 管道的功能
- 使用管道组合命令

6.1　标准文件

【问题的提出】

Geecy 软件开发公司的开发人员 Tom 遇到一个问题：Tom 经常需要把两个文本文件合并为一个，如果使用 vi 合并，其操作比较繁琐。Tom 需要寻找一种简便的合并方法。

【问题分析】

使用 vi 可以合并两个文本文件，但操作相当繁琐。首先，需要打开第一个文件，然后移动光标到文件的最后，再使用命令“ :r ”读入第 2 个文件，最后保存并退 vi。

确实是足够繁琐。

Tom 也想到另一种方法：使用他熟悉的 C 语言编写一个简单的程序，该程序运行时候从命令行接收两个文件名作参数，然后把第 1 个文件的内容添加到第 2 个文件，最后关闭文件并退出。

既然是这么简单的程序，Linux 会提供吗？因此，Tom 还没有开始动手编写该 C 语言程序。

6.1.1　标准文件简介

Linux 系统把所有的设备当作文件来管理，每个设备都有相应的文件名。例如，使用 ls 命令：

```
tom@geecy:~$ ls -l /dev↙
crw-rw----    1 root     video     10, 175 Jan 30  2005 agpgart
drwxr-xr-x    2 root     root         4096 Jan 30  2005 ataraid
crw-------    1 root     root      10,   3 Mar 15  2002 atibm
crw-rw----    1 root     audio     14,   4 Mar 15  2002 audio
(省略后面内容)
```

根据表 3-1 的文件类型符号说明可以知道，“ b ”和“ c ”都表明该文件是一个设备文件。

在 Linux 系统中，输入输出设备也被当作是一个文件来看待。使用 ls 命令继续查看：

```
tom@geecy:~$ ls -l /dev/std*↙
```

```
lrwxrwxrwx    1 root     root            4 Jan 30  2005 /dev/stderr -> fd/2
lrwxrwxrwx    1 root     root            4 Jan 30  2005 /dev/stdin -> fd/0
lrwxrwxrwx    1 root     root            4 Jan 30  2005 /dev/stdout -> fd/1
```

其中，文件“ /dev/stdin ”称为标准输入(Standard Input)文件，简称 stdin；文件“ /dev/stdout ”称为标准输出(Standard Output)文件，简称 stdout；文件“ /dev/stdin ”称为标准错误(Standard Error)文件，简称 stderr。

这三个文件分别链接指向到目录 fd 下的三个以数字命名的文件“ 0 ”、“ 1 ”和“ 2 ”。

stdout 和 stderr 都是关联到显示器，而 stdin 关联到键盘。

如果一个程序需要显示内容到显示器，那么只需要把内容输出到 stdout。如果一个程序需要从键盘上读入内容，那么只需要读取 stdin。

例如，ls 命令的输出结果默认与标准输出文件关联，因此用户使用 ls 命令的输出结果是显示在屏幕上的。某些需要与用户交互的程序，比如你用户使用的 Shell（诸如 sh、bash 等），是从标准输入文件读取你输入的命令的。

程序所产生出错信息会输出到 stderr。例如，cat 命令后接不存在的文件名，则会产生错误信息输出到显示器。

系统对这三个标准文件分别赋予了一个整数，称为文件描述符(File Descriptor)。其中 stdin 对应的文件描述符为 0，stdout 对应的文件描述符为 1，stderr 对应的文件描述符为 2。

6.1.2 文件重定向

由于使用标准文件，因此应用程序的使用可以变得很灵活。例如，如果 ls 命令需要输出结果到屏幕，那么只需要把结果送到 stdout。因为 stdout 是作为一个文件被看待，所以用户可以想办法通过把文件 stdout 换成另一个指定的普通文件来“ 欺骗 ”ls 命令，这样结果就被送到文件去保存，而不送去屏幕显示。这就是文件的重定向(Redirect)。

根据被重定向的文件的不同，分为三种重定向。

如果 stdout 被重定向，那么称为输出重定向。符号“ > ”表示输出重定向的操作。例如：

```
tom@geecy:~$ ls -l /sbin  >  /tmp/log.txt↙
```

命令“ ls -l /bin ”的输出结果被重定向保存到文件“ /tmp/log.txt ”，不再输出到屏幕。

符号“ >> ”和“ > ”都表示输出重定向，但是有区别：

- “ > ” 表示把左边命令的结果重定向到右边的文件，如果文件已经存在，则覆盖原有的文件，如果文件不存在，则创建新文件。
- “ >> ”表示把左边命令的结果重定向到右边的文件，如果文件已经存在，则添加内容到该文件的末尾，如果文件不存在，则创建新文件。

如果 stdin 被重定向，那么称为输入重定向。符号“ < ”表示输入重定向的操作。

不带参数的 cat 命令默认从标准输入文件，即键盘获取内容，然后原样输出到标准输出文件，即显示器。例如：

```
tom@geecy:~$ cat↙
hello↙
hello
a test for cat command↙
a test for cat command
<Ctrl>+d
```

【注意】<Ctrl>+d 表示强行终止命令的执行。

用户可以使用输入重定向，实现命令“ cat /etc/hosts ”的同样功能：

```
tom@geecy:~$ cat < /etc/hosts↙
```

如果 stderr 被重定向，那么称为错误重定向。错误重定向使用符号“ 2> ”和“ 2>> ”。

使用“ 2 ”的原因是标准错误文件的文件描述符是 2。事实上，标准输出文件可以使用符号“ 1> ”和“ 1>> ”，同时，使用其简单形式“ > ”和“ >> ”也兼容；标准输入文件可以使用符号“ 0< ”，同时，使用其简单形式“ < ”也兼容。

但是，符号“ 2> ”和“ 2>> ”中的 2 不能省略（如果省略，就与输出重定向产生冲突）。

例如：

```
tom@geecy:~$ cat /etc/host↙
cat: /etc/host: No such file or directory
```

因为文件“ /etc/host ”不存在，所以显示出错信息。

下面使用错误重定向的技术，避免出错信息输出到屏幕。

```
tom@geecy:~$ cat /etc/host 2> log.txt↙
tom@geecy:~$ cat log.txt↙
cat: /etc/host: No such file or directory
tom@geecy:~$ cat /etc/abc 2>> log.txt↙
tom@geecy:~$ cat log.txt↙
cat: /etc/host: No such file or directory
cat: /etc/abc: No such file or directory
tom@geecy:~$ cat /etc/abcdef 2> log.txt↙
tom@geecy:~$ cat log.txt↙
cat: /etc/abcdef: No such file or directory
```

【解决方案】

Tom 可以输出重定向技术解决该问题。假设两个文件是 file1.txt 和 file2.txt，合并后保存在 file3.txt。则操作如下：

```
tom@geecy:~$ cat file1.txt > file3.txt↙
tom@geecy:~$ cat file2.txt >> file3.txt↙
```

6.2 管道

【问题的提出】

Geecy 软件开发公司的系统管理员 Ray 需要经常检查目录/sbin 下的可执行文件，防止自己或其他用户在执行某些命令时候有意或无意对这些重要的可执行文件产生破坏。Ray 想到一种方法，就是把目录/sbin 下的每个文件名和文件大小记录到一个文本文件，以后可以作对比参考。因为可执行文件如果受破坏，一般都会影响到文件的大小。

但目录/sbin 下文件很多，不可以逐一手工记录。Ray 希望能够找到实现该任务的命令。

【问题分析】

Linux 系统提供了丰富的命令供用户使用。当没有一个命令能够完全满足用户的需要时，就应该考虑使用若干个命令的组合来实现。Linux 系统的设计思想正是如此，系统提供了大量短小精悍的命令，用户可以利用管道和重定向的原理来组合使用这些命令，以实现复杂的任务。

6.2.1 管道的功能

用户使用下面的命令一般都会遇到麻烦：

```
ray@geecy:~$ ls -l /sbin↙
```

该命令的输出结果远远超出了一个屏幕的行数。当然，用户可以使用“硬”的方法实现（请参考“1.4.2 通过控制台使用 Linux”）。也可以使用“软”的方法，即前面介绍的重定向方法：

```
ray@geecy:~$ ls -l /sbin > /tmp/ls.log↙
ray@geecy:~$ less /tmp/ls.log↙
```

先把结果重定向到一个临时文件，再使用有分屏功能的 less 命令查看，从而实现全部结果的浏览。

这两种方法都有一个共同的缺陷：操作不方便。

事实上，用户可以使用管道(Pipe)功能完美地解决这类问题。

管道功能在本质上是属于重定向功能，只不过它没有要求用户指定一个文件名，而是利用内存实现文件信息的保存。由于是使用内存代替磁盘文件来存放临时信息，除了简便，还带来一个附加的好处——加快执行速度。

管道功能使用符号“|”实现。在键盘上它与符号“\”位于同一个键。例如，上面例子改为使用管道实现：

```
ray@geecy:~$ ls -l /sbin | less↙
```

管道的功能是把左边命令的输出重定向，传送给右边的命令作为输入；同时把右边命令的输入重定向，以左边命令的输出结果作为输入。

6.2.2 使用管道组合命令

管道具有把多个命令从左到右“串联”起来的能力。这些被串联的命令所完成任务都有一个共同点：把从左边输入重定向过来的内容经过一些“过滤”处理，然后重定向输入到下一个命令作为输入。如此不断，就可以形成一条串联起来的“管子”。

不是所有的命令都适合使用在这条长“管子”的中间。下面介绍几个被认为是比较适合使用于管道的命令。这些命令通常被称为“过滤器”(Filter)。

常见的过滤器有 wc、cut、tr、grep 等。

■ wc

wc 命令具有行数统计(Line Count)、单词统计(Word Count) 和字符统计(Character Count)的功能。

例如：

```
ray@geecy:~$ wc↙
this is line one.
this is line two.
this is line three.
<Ctrl>+d
      3      12      56
```

运行单独的 wc 命令，将从标准输入文件（即键盘）接收输入信息，然后统计这些信息中共有多少行、多少个单词以及多少个字符，直到用户使用组合键盘<Ctrl>+d 终止输入为止。最后打印统计信息到标准输出设备（即显示器)。

本例子中，wc 命令统计的信息是：共 3 行、12 个单词以及 56 个字符。

基于 wc 命令的这些特点，很容易应用于管道功能。例如，统计文件 hello.c 的行数、单词数以及字符数：

```
ray@geecy:~$ cat hello.c↙
#include <stdio.h>
int main()
{
        char message_str[]="Hello,world! I am using vi...";
        char format_str[]="%s\n";
        printf(format_str,message_str);
        return 0;
}
ray@geecy:~$ cat hello.c | wc↙
      8      17     158
```

统计结果是 8 行、17 个单词以及 158 个字符。

wc 命令统计单词的依据是以空格为分隔作为一个单词，而不是通常意义上的单词。wc 命令统计字符数，其实就是文本文件的大小：

```
ray@geecy:~$ ls -l hello.c↙
-rw-r--r--    1 ray      ray           158 Jan 26 09:49 hello.c
```

wc 命令还提供选项让用户自行选择需要统计的内容：

```
ray@geecy:~$ cat hello.c | wc -l↙
      8
```

选项“ -l ”指出需要统计行数。

```
ray@geecy:~$ cat hello.c | wc -w↙
      17
```

选项“ -w ”指出需要统计单词数。

```
ray@geecy:~$ cat hello.c | wc -c↙
      158
```

选项“ -c ”指出需要统计字符数。

当然，这些选项还可以组合使用：

```
ray@geecy:~$ cat hello.c | wc -lc↙
      8         158
```

选项“ -lc ”指出需要统计行数和字符数。

■ cut

cut 命令具有从指定文件的逐行提取特定的列的内容输出到标准输出文件的功能。cut 命令的特点是以行为单位工作。

cut 命令可以按照两种方式来提取内容。

一种是指定某个字符作为定界符，把每行内容分为若干列，然后再指定提取某些列，从而形成对输入内容的过滤。

例如，文件“ /etc/passwd ”的内容如下：

```
ray@geecy:~$ cat /etc/passwd↙
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:100:sync:/bin:/bin/sync
（省略后面内容）
```

cut 命令重定向输入从管道传过来的信息，使用选项“ -d ”指定字符“ : ”作为定界符，把每行内容分为若干列，然后再使用选项“ -f ”指定提取每行的第 1 列的内容。

```
ray@geecy:~$ cat /etc/passwd | cut -f 1 -d ":"↙
root
daemon
bin
sys
sync
（省略后面内容）
```

该命令的结果是显示了每行的第 1 个数据域内容，即用户名。

选项“ -f ”指定提取的列可以很灵活。例如：

```
ray@geecy:~$ cat /etc/passwd | cut -f 1-3 -d ":"↙
root:x:0
daemon:x:1
bin:x:2
sys:x:3
sync:x:4
（省略后面内容）
```

选项“ -f 1-3 ”表明需要提取第 1 列至第 3 列的内容。而选项“ -f 1,3 ”则表明需要提取第 1 列和第 3 列的内容：

```
ray@geecy:~$ cat /etc/passwd | cut -f 1,3 -d ":"↙
root:0
daemon:1
bin:2
sys:3
sync:4
（省略后面内容）
```

又或者是组合使用“ - ”和“ , ”:

```
ray@geecy:~$ cat /etc/passwd | cut -f 1-3,7 -d ":"↙
root:x:0:/bin/bash
daemon:x:1:/bin/sh
bin:x:2:/bin/sh
sys:x:3:/bin/sh
sync:x:4:/bin/sync
（省略后面内容）
```

选项“ -f 1-3,7 ”表明需要提取第 1 列至第 3 列的内容以及第 7 列的内容。

另一种方式是指定提取每行的某些位置的字符，相当于把每行的内容以一个字符为单位分为若干列，然后再指定提取某些列，从而形成对输入内容的过滤。

例如：

```
ray@geecy:~$ cat /etc/passwd | cut -c 1↙
r
d
b
s
s（省略后面内容）
```

cut 命令重定向输入从管道传过来的信息，把每行的内容以一个字符为单位分为若干列，然后使用选项“ -c 1 ”指定提取每行的第 1 个字符的内容

该命令的结果是显示了每行的第 1 个字符的内容，即用户名的第 1 个字符。

选项“ -c ”与选项“ -f ”的提取格式类似。例如：

```
ray@geecy:~$ cat /etc/passwd | cut -c 1-3↙
roo
dae
bin
sys
syn
（省略后面内容）
```

选项“ -c 1-3 ”表明需要提取第 1 至第 3 个字符的内容。而选项“ -c 1,3 ”则表明需要提取第 1 个和第 3 个字符：

```
ray@geecy:~$ cat /etc/passwd | cut -c 1,3↙
ro
de
bn
ss
sn
（省略后面内容）
```

■ tr

tr 命令有两大功能：一是能够把从标准输入文件读入的一个字符集合翻译成另一个字符集合然后输出到标准输出文件；二是能够把连续几个相同的字符压缩为一个字符。

例如：

```
ray@geecy:~$ cat /etc/passwd | tr  ":"  " "↙
root x 0 0 root /root /bin/bash
daemon x 1 1 daemon /usr/sbin /bin/sh
bin x 2 2 bin /bin /bin/sh
sys x 3 3 sys /dev /bin/sh
sync x 4 100 sync /bin /bin/sync
（省略后面内容）
```

tr 命令重定向输入从管道传过来的信息，然后使用第 2 个参数指定的字符“ ”（空格）取代第 1 个参数指定字符“ : ”，过滤后的内容最后输出到屏幕上。

下面的例子是把字符“ : ”过滤为“ , ”然后输出到屏幕上：

```
ray@geecy:~$ cat /etc/passwd | tr  ":"  ","↙
root,x,0,0,root,/root,/bin/bash
daemon,x,1,1,daemon,/usr/sbin,/bin/sh
bin,x,2,2,bin,/bin,/bin/sh
sys,x,3,3,sys,/dev,/bin/sh
sync,x,4,100,sync,/bin,/bin/sync
（省略后面内容）
```

tr 命令不但可以对一个字符进行过滤，而且可以对整个给定的字符集合过滤。例如：

```
ray@geecy:~$ cat /etc/passwd | tr "[a-z]" "[A-Z]"↙
ROOT:X:0:0:ROOT:/ROOT:/BIN/BASH
DAEMON:X:1:1:DAEMON:/USR/SBIN:/BIN/SH
BIN:X:2:2:BIN:/BIN:/BIN/SH
SYS:X:3:3:SYS:/DEV:/BIN/SH
SYNC:X:4:100:SYNC:/BIN:/BIN/SYNC
（省略后面内容）
```

tr 命令重定向输入从管道传过来的信息，然后进行过滤。凡是属于集合“ [a-z] ”的字符，均使用集合“ [A-Z] ”中相应的字符替换。过滤后的内容最后输出到屏幕上。

本例子的功能其实就是相当于把文件内容中的小写字母转换为大写。

tr 命令的压缩功能如下例子所示：

```
ray@geecy:~$ who↙
root     tty1         Jan 26 14:05
ray      pts/0        Jan 26 13:16 (192.168.1.2)
ray@geecy:~$ who | tr -s " "↙
root tty1 Jan 26 14:05
ray pts/0 Jan 26 13:16 (192.168.1.2)
```

tr 命令重定向输入从管道传过来的信息，然后使用选项“ -s ” 把多个空格（ ” ”）压缩(Squeeze)成一个，最后输出到屏幕。

■ grep

grep 命令能够从给定的文件中查找包含某种模式的行。该模式由用户使用正则表达式指定。关于模式和正则表达式请参考“5.1.3 vi 的文本搜索”。

例如：

```
ray@geecy:~$ grep "root" /etc/passwd↙
root:x:0:0:root:/root:/bin/bash
```

grep 命令的第一个参数代表需要匹配的模式，第二个参数指出搜索的文件。上述命令的功能是在文件“/etc/passwd”中搜索含有字符串“root”的行，并把这些行的整行内容显示到屏幕上。

再如：

```
ray@geecy:~$ grep "main"  hello.c↙
int main()
```

如果不指定文件名，那么 grep 命令从标准输入文件（键盘）读取信息。例如：

```
ray@geecy:~$ grep "day"↙
date↙
day↙
day
week↙
<Ctrl>+d
```

该命令从标准输入文件查找包含字符串“day”的行并输出到屏幕。

【解决方案】

Ray 终于找到了实现方法。就是利用 cut 命令抽取所需要的列（文件大小和文件名称）。在抽取之前还要压缩每行的空格，否则无法正确计算需要抽取的列数。

Ray 使用管道组合命令来实现：

```
ray@geecy:~$ ls -l /sbin | tr -s " " | cut -d " " -f 5,9 > bin_log.txt↙
ray@geecy:~$ less bin_log.txt↙
43485 MAKEDEV
4224 activate
14408 badblocks
7372 blockdev
45192 cfdisk
48168 debugfs
（省略后面内容）
```

以后可以使用 less 命令方便地查看。

6.3 练 习

1、 Cell 宽带数据公司公司的客户材料保存在 Customer 文件中。示例数据如下：

```
"00001", "Huang Dafeng", "152 Xingang Xi", "Guangdong","Guangzhou"
"00002", "Luo Jia", "246 Huaqiang Bei", "Guangdong", "Shenzhen"
"00003", "Zhang Chan", "246 Huaqiang Bei", "Guangdong", "Shenzhen"
… …
… …
```

字段之间的定界符是“ , ”，字段的含义依次是代码、姓名、地址、省份和城市。Tenny 需要：

- 写一个命令，统计客户的数量（即文件的行数）。
- 写一个命令，只显示生活在城市 Guangzhou 的客户的人数。
- 写一个命令，只显示生活在城市 Guangzhou 的客户的姓名。
- 写一个命令，以大写形式显示该文件的内容，并以分屏的形式输出。

2、 使用管道组合一个命令，实现显示当前目录下其文件拥有者具有读和写权限的普通文件的文件名。（注意：只是显示文件名）

第 7 章　Shell 脚本设计

本章提要

- Shell 脚本简介
- 使用变量
- 命令替换
- 数值运算
- 流程控制

7.1　Shell 脚本简介

【问题的提出】

Geecy 软件开发公司的开发人员 Tom 在调试程序的过程中遇到了一个令人烦恼的问题：在运行一个用 C 语言编写的应用程序的时候，该程序需要读取一个包含当前时间值的文本文件，该文件的格式必须为：第 1 行是“月-日-年”，第 2 行是“时:分”。

但 date 命令的默认显示格式是“星期 月 日 时:分:秒 时区 年”。

因此，Tom 每次运行程序前，都要手工修改该数据文件，即先运行 date 命令，再修改数据文件，最后运行该应用程序。而且从运行 date 命令到运行该应用程序最长时间不应该超过 1 分钟，否则就要重新修改数据文件。

【问题分析】

有两种传统的方法可以完成对该数据文件的修改：

■ **使用文本编辑器**

首先运行 date 命令，记下当前时间，然后使用 vi 打开该数据文件，修改时间信息并保存退出。最后，要立即运行该应用程序。

此过程会耗费一定的时间，如果系统时钟的分钟数发生改变（到了下一分钟），那么将影响该应用程序的调试。此操作过程必须重做。

■ **使用组合命令**

可以充分利用重定向和管道的功能实现。步骤如下（假设数据文件是~/data.txt）：

```
tom@geecy:~$ date  | cut  -f2,3,6  -d  " "  |  tr  " "  "-"  >  data.txt↙
tom@geecy:~$ date  | cut  -f4  -d  " " | cut -f1,2 -d ":"  >>  data.txt↙
```

然后就可以运行调试该应用程序。用户也可以检验：

```
tom@geecy:~$ cat data.txt↙
Jan-30-2005
09:51:59
```

虽然第二种方法有所改进，但还是需要用户每次重复输入两个很长的命令。

7.1.1 认识 Shell 脚本

事实上，用户可以把若干个命令保存到一个文本文件，然后使用该文件的名字一次执行所有的命令。这些命令将逐行执行。这样的文本文件称为 Shell 脚本(Shell Script)。以下简称脚本。

脚本是一组命令的集合。凡是能够在 shell 提示符下直接执行的命令，都可以在脚本中使用。脚本中还可以包含一些不能在 shell 提示符下直接执行的语句，这些语句必须在脚本中使用才有效。

7.1.2 编写简单脚本

用户可以使用 vi 编辑一个新文件 fist_script：

```
tom@geecy:~$ vi  first_script↙
```

输入以下两行内容：

```
ls -F /
date
```

然后保存文件并退出 vi。

一个很简单的脚本 first_script 已经建立了起来。

7.1.3 运行脚本

用户可以在命令提示符下使用某个 Shell 执行该脚本。例如：

```
tom@geecy:~$ bash  first_script↙
```

在用户当前使用的 Shell(不一定是 bash)下首先运行 bash 作为一个子 Shell,该子 Shell 读入脚本文件 first_script 然后逐行地执行脚本中的命令并依次输出结果：

```
bin/    dev/     home/    lost+found/  proc/  tmp/  vmlinuz@
boot/   etc/     initrd/  mnt/         root/  usr/
cdrom/  floppy/  lib/     opt/         sbin/  var/
Sun Jan 30 10:16:13 CST 2005
```

一旦脚本文件 first_script 中的命令执行完毕，该临时的子 Shell 也自动结束运行。返回到用户原来所使用的 Shell：

```
tom@geecy:~$
```

对于大部分的命令，多数类型的 Shell 一般都可正常执行，即兼容性比较好。在本例中，用户可以使用 sh 代替 bash 运行：

```
tom@geecy:~$ sh  first_script↙
```

另一种运行脚本的方法则更为常用，具体操作是：

用 vi 修改脚本 first_script，在文件的开头增加一行内容指定一个 shell：

```
#!/bin/bash
ls -F /
date
```

“#!”必须是整个文件的第一行的前两个字符，后接 shell 的绝对路径。

然后保存文件并退出 vi。再修改脚本 first_script 的文件访问权限(FAP)，增加文件拥有者的执行权限：

```
tom@geecy:~$ chmod u+x first_script↙
tom@geecy:~$ ls -l first_script↙
-rwxr--r--    1 tom      tom            26 Jan 30 10:35 first_script
```

最后可以像使用系统命令一样直接运行脚本 first_script：

```
tom@geecy:~$ ./first_script↙
```

【注意】由于 Linux 系统默认配置的搜索路径 PATH 不包含当前工作目录，因此这里需要添加“./”指定确切的路径。当然，用户也可以使用绝对路径，例如“/home/tom/first_script”。

用户可以从文件/etc/shells 获知系统中所有可用的 shell 以及其绝对路径：

```
tom@geecy:~$ cat /etc/shells↙
```

7.1.4 适当注释脚本

为了增加可读性或是别的原因，用户可以在脚本中以行为单位插入注释(Comment)。所注释的行使用字符“#”开头，当 Shell 遇到“#”开头的行将会忽略该行。例如：

```
#!/bin/bash

#This will list the content of directory /bin in classify format
ls -F /

#This will list the current time
date
```

适当地插入注释和空行都能够增加脚本程序的可读性。

【解决方案】

Tom 最后编写一个脚本程序把那个恼人的问题解决：

首先使用 vi 编写创建新文件 update：

```
tom@geecy:~$ vi  update↙
```

输入如下内容：

```
#!/bin/bash
# Add a formatted string of date to data.txt
date  | cut  -f2,3,6  -d  " "  |  tr  " "  "-"  >  data.txt

# Add a formatted string of time to data.txt
date  | cut  -f4  -d  " " | cut -f1,2 -d ":"  >>  data.txt

# Display the content of data.txt
cat  data.txt
```

然后保存并退出。再给脚本 update 加上执行权限：

```
tom@geecy:~$ chmod  u+x  update↙
```

以后，Tom 就可以随时通过执行脚本 update 轻松快速地生成测试数据文件 data.txt：

```
tom@geecy:~$ ./update↙
Jan-30-2005
12:06
```

7.2 使用变量

【问题的提出】

Geecy 软件开发公司的开发人员 Tom 已经开始使用自己开发的脚本 update。但随之而来又产生一个新问题：很多时候需要产生不同文件名的数据文件，也就是说，文件名不一定是固定的“ data.txt ”!

【问题分析】

这个问题本身不难。Tom 立刻就想到了解决的方法：

例如，如果要求产生的数据文件名字为“ new.txt ”，则可以：

```
tom@geecy:~$ ./update↙
tom@geecy:~$ mv  data.txt  new.txt↙
```

也就是说，每次执行脚本 update 之后，再运行一个文件重命名的命令。

更好的解决方法是使用变量。

7.2.1 变量创建和引用

使用 Bash Shell，用户不需要显式地声明变量。你可以在赋值给一个新变量的同时创建变量。创建变量并赋值的语法是：

<Variable>=<Value>

其中：

(1) <Variable>表示将要创建的变量名。

(2) <Value>表示赋给变量的值。

(3) 赋值运算符“ = ”的两边必须没有空白。

例如：

```
tom@geecy:~$ author=Jack↙
tom@geecy:~$ title="C++  Programming"↙
tom@geecy:~$ price=28↙
```

创建了三个变量“ author ”、“ title ”和“ price ”，并分别赋值。

声明变量的时候，用户不需要指定变量的类型。Shell 脚本中的变量都被当作是字符串。如果所赋值的内容没有空格，可以不使用引号。

当需要引用一个变量的内容，需要使用符号“ $ ”。为了不引起歧义，一般还需要配合花括号“ { } ”。如果不涉及到字符串的连接，那么“ { } ”是可选的。

例如：

```
tom@geecy:~$ FirstName="Linus  "↙
tom@geecy:~$ LastName=Torvalds↙
tom@geecy:~$ LinuxAuthor=$FirstName↙
tom@geecy:~$ FullName=${FirstName}${LastName}↙
```

首先分别创建名字为 FirstName 和 LastName 的两个变量并赋值；

然后创建名字为 LinuxAuthor 的变量并赋值为变量 FirstName 的内容；

最后创建名字为 FullName 的变量并赋值为变量 FirstName 与 LastName 连接后的内容。

7.2.2 变量的读入与输出

■ **输出变量内容**

如果需要输出内容到屏幕，用户可以使用 echo 命令。使用 echo 命令的语法是：

```
echo  [-n]  String
```

其中：

(1) String 表示将要输出的字符串。

(2) 选项-n 表示输出 String 后不输出回车换行符。

例如，使用 vi 创建一个脚本：

```
#!/bin/bash
echo  "This is the first line."
echo  -n  "This is the second line."
echo  "This is also the second line."
echo  "This is the third line."
```

运行后的结果将是：

```
This is the first line.
This is the second line.This is also the second line.
This is the third line.
```

由于第二个 echo 命令使用的选项“ -n ”而不输出回车换行符，因此本应是在第三行显示的内容改为在第二行的末尾接着输出。

在默认方式下，echo 命令输出字符串后并在后面输出一个回车换行符，使光标移动到下一行的行首，等待后面的输出。如果选用了选项“ -n ”，则可以在 echo 命令输出字符串后把光标保持在该字符串的末尾，等待后面的输出。

用户也可以先保存一个字符串内容到变量，然后在需要的地方输出该变量内容。

例如，使用 vi 把上述脚本修改为：

```
#!/bin/bash
str1="This is the first line."
str2="This is the second line."
```

```
str3="This is also the second line."
str4="This is the third line."
echo  $str1
echo  -n  $str2
echo  $str3
echo  $str4
```

这两个脚本的输出结果相同。

■ **读入内容到变量**

除了直接赋值给变量，用户也可以从键盘中读入一个值赋给变量，然后在 shell 脚本的后面根据需要使用该变量。使用 read 命令从键盘读入内容到变量。

当脚本解释执行到 read 命令的时候，将会暂停并等待用户的键盘输入。当用户输入内容并按回车键后，后面的语句将继续执行。

由于 read 命令不带提示信息，因此一般需要预先用 echo 命令输出一些必要的提示信息。

例如，使用 vi 创建一个脚本：

```
#!/bin/bash
echo  "What is your name?"
read  UserName
echo  "Hello "${UserName}", nice to meet you! "
```

运行后的结果将是：

```
What is your name?
Huang Dafeng↙
Hello Huang Dafeng, nice to meet you!
```

首先由 echo 命令给出提示信息“What is your name?”；然后执行 read 命令，并等待用户的键盘输入，当用户输入内容（可以包含空格）并回车后，用户所输入内容被保存到名为 UserName 的变量中；最后“Hello ”、“Huang Dafeng”和“, nice to meet you!”三个字符串连接并输出到屏幕。

另一个例子是：

```
#!/bin/bash
echo  -n  "Enter your name: "
read  Name
echo  -n  "Enter your phone number: "
read  Phone
echo  "$Name:$Phone"  >>  file.txt
```

该脚本将从键盘接收用户输入的名字和电话号码，然后以冒号分隔的形式添加一行到文件 file.txt 的末尾。

【解决方案】

Tom 找到了更简单的解决方法：

使用 vi 把脚本 update 修改为：

```
#!/bin/bash
# Receive a file name
echo  -n  "Input a file name: "
read  FileName

# Add a formatted string of date to file
date  | cut  -f6,2,3  -d  " "  |  tr  " "  "-"  >  $FileName

# Add a formatted string of time to data.txt
date  | cut  -f4  -d  " " | cut -f1,2 -d ":"  >>  $FileName

# Display the content of file
cat  $FileName
```

保存脚本并运行：

```
tom@geecy:~$ ./update↙
Input a file name: another↙
Jan-30-2005
12:06
```

验证文件是否创建：

```
tom@geecy:~$ cat another↙
Jan-30-2005
12:06
```

从此，Tom 可以使用脚本 update 轻松创建不同名字的数据文件。

【问题的提出】

Geecy 软件开发公司的开发人员 Tom 觉得当前所使用的默认命令提示符“ tom@geecy:~$ ”不是很习惯。Tom 认为信息“ tom@geecy ”对他而言是多余的，他更习惯于使用 MS-DOS 风格的提示符，例如“ C:\Dos6.22> ”。

现在，Tom 希望把默认命令提示符修改包含当前时间和当前工作目录的绝对路径的形式：

```
02:06pm /usr/bin >
```

【问题分析】

Shell 中的变量分为两种：一是用户自定义的变量，例如：

```
tom@geecy:~$ author=Jack↙
```

这种变量只对用户有意义，不会对系统产生影响；

另一种称为系统环境变量，这些变量由系统创建和赋值。系统环境变量中有一些变量与用户紧密相关，用户甚至可以修改其值，以达到改变系统的某些行为的目的。例如，用户如果需要修改系统默认的命令提示符，就需要分辨出究竟是哪个系统变量控制着系统默认命令提示符的表现形式，然后修改该系统变量，从而实现命令提示符的修改。

7.2.3 系统环境变量

当一个用户登录到 Linux 系统，Linux 系统就自动为该用户运行一个 Shell，该 Shell 称为登录 Shell(Login Shell)。登录到 Linux 系统后，用户也可以输入某个 Shell 的可执行文件直接运行一个新的 Shell，该 Shell 称为非登录 Shell。

登录 Shell 运行成功后，会创建一些由系统预先定义的变量，成为环境变量(Environment Variable)。

用户可以使用 env 命令查看一些跟用户有直接关系的最常用的环境变量：

```
tom@geecy:~$ env↙
PWD=/home/tom
HZ=100
PS1=\u@\h:\w\$
USER=geecy
MAIL=/var/mail/tom
LOGNAME=tom
SHLVL=1
SHELL=/bin/bash
TERM=xterm
HOME=/home/ray
PATH=/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
_=/usr/bin/env
```

而使用 set 命令则可以查看更多的环境变量：

```
tom@geecy:~$ set↙
```

■ LOGNAME 变量

该变量保存了用户的登录用户名(Login Name)。用户可以单独查看该变量的内容:

```
tom@geecy:~$ echo  $LOGNAME↙
```

■ SHELL 变量

该变量保存了用户的登录 shell。用户可以单独查看该变量的内容:

```
tom@geecy:~$ echo  $SHELL↙
```

■ HOME 变量

Linux 系统中每个用户都有一个个人目录。当用户登录成功后，系统自动把该用户的个人目录设置为当前工作目录。

用户的个人目录的绝对路径保存在 HOME 变量中。用户可以单独查看该变量的内容:

```
tom@geecy:~$ echo  $HOME↙
```

■ PATH 变量

该变量包含了一系列用冒号分隔的目录名。当用户发出一个命令，如果不是使用绝对路径的话，系统将会从 PATH 变量中依次提取目录名，并从这些目录中搜索用户所发出命令的可执行文件，找到后就执行该命令。如果搜索了 PATH 变量保存的所有目录后都没有找到相应的可执行文件，则系统输出出错信息。用户可以单独查看该变量的内容：

```
tom@geecy:~$ echo  $PATH↙
```

还可以修改 PATH 变量的内容以适应用户的要求。例如：

```
tom@geecy:~$ PATH=~/bin:$PATH↙
```

将在原来的搜索路径之前增加一个“ /home/tom/bin ”的目录。如果用户个人使用的可执行文件保存在该目录下，就可以直接运行，而无需增加“ ./ ”之类的前缀。

■ SHLVL 变量

该变量保存了当前工作 shell 的层次(Shell Level)。在 Linux 中，用户可以在某个 shell 中创建一个子 shell，然后在该子 shell 中再创建子 shell，依次类推。用户有时候需要了解当前 shell 所处于哪一个层次上。登录 shell 的层次是 1。当创建了一个子 shell 后，其层次值递增 1。

例如：

```
tom@geecy:~$ echo  $SHLVL↙
1
tom@geecy:~$ bash↙
tom@geecy:~$ echo  $SHLVL↙
2
```

```
tom@geecy:~$ bash↙
tom@geecy:~$ echo  $SHLVL↙
3
tom@geecy:~$ exit↙
tom@geecy:~$ echo  $SHLVL↙
2
```

■ PS1 变量

PS1(Prompt String 1)变量保存了登录 shell 的提示符。在 Linux 中，Bash Shell 默认的提示符如下所示：

```
tom@geecy:~$
```

PS1 变量的默认配置为：

```
tom@geecy:~$ echo  $PS1↙
\u@\h:\w\$
```

其中“ \u ”表示登录用户名，“ \h ”表示主机名称，“ \w ”表示用户当前工作目录的绝对路径。符号“ $ ”本身具有特殊含义，即用来引用变量，在此“ \$ ”表示取消其特殊含义，成为一个普通字符。前面没有“ \ ”的字符没有特别的含义，因此，字符“ @ ”、“ : ”和“ \$ ”是普通字符。

综合起来，Bash 默认提示符的风格就是“ 用户名@主机名:当前工作目录绝对路径$ ”。

修改 PS1 变量的内容可以改变命令提示符的风格：

```
tom@geecy:~$ PS1="\w >"↙
~ >
```

提示符的风格被修改为“ 当前工作目录绝对路径 > ”。

7.2.4 变量的作用域

■ 局部变量

在某个 shell 中创建的变量，通常是一个局部变量，即它只能在所创建的 shell 中有效。一般情况下某个 shell 与它的子 shell 中的变量是各自相对独立的，互相不受影响。也就是说，变量的作用域一般就是所创建它的当前 shell。这种变量称为局部变量，或者称为本地变量(Local Variable)。

例如，首先在当前的 shell 中创建一个变量 COLOR，赋值 blue：

```
tom@geecy:~$ COLOR=blue↙
tom@geecy:~$ echo  $COLOR↙
blue
```

然后，创建一个新的 shell，并以该子 shell 作为当前工作的 shell，在子 shell 中，显示变量 COLOR 的内容，是空白，说明了变量 COLOR 不能自动传递到子 shell 使用：

```
tom@geecy:~$ bash↙
```

```
tom@geecy:~$ echo  $COLOR↙

```

接着，在子 shell 中创建一个变量 COLOR，赋值 green。显示变量 COLOR 的内容，是 green：

```
tom@geecy:~$ COLOR=green↙
tom@geecy:~$ echo  $COLOR↙
green
```

最后，退出当前的子 shell，返回原来的 shell。显示变量 COLOR 的内容，仍然是 blue，说明了它不受子 shell 的影响：

```
tom@geecy:~$ exit↙
tom@geecy:~$ echo  $COLOR↙
blue
```

■ **全局变量**

有些时候，用户需要使用全局变量(Global Variable)。在 shell 脚本中，全局变量能够被复制一个副本传递到子 shell 使用，子 shell 可以修改该副本的内容。

由于是副本，因此一旦退出该子 shell 回到原来的 shell 后，该变量的内容不受子 shell 的影响。用户可以使用 export 命令把一个普通的局部变量变成全局变量。

例如，首先在当前的 shell 中创建一个变量 COLOR，赋值 blue，再使用 export 命令移出变量 COLOR，使其成为全局变量。：

```
tom@geecy:~$ COLOR=blue↙
tom@geecy:~$ export  COLOR↙
tom@geecy:~$ echo  $COLOR↙
blue
```

【注意】export 命令后面的参数不需要使用符号“$”引用。

然后，创建一个新的 shell，并以该子 shell 作为当前工作的 shell，在子 shell 中，显示变量 COLOR 的内容，也是 blue，说明了变量 COLOR 已经传递到子 shell：

```
tom@geecy:~$ bash↙
tom@geecy:~$ echo  $COLOR↙
blue
```

接着，在子 shell 中修改变量 COLOR 的内容为 green。显示变量 COLOR 的内容，是 green：

```
tom@geecy:~$ COLOR=green↙
tom@geecy:~$ echo  $COLOR↙
green
```

最后，退出当前的子 shell，返回原来的 shell。显示变量 COLOR 的内容，仍然是 blue，说明了它不受子 shell 的影响：

```
tom@geecy:~$ exit↙
tom@geecy:~$ echo  $COLOR↙
blue
```

【解决方案】

Tom 已经了解到，系统环境变量 PS1 控制系统默认命令提示符的表现形式，“ \w ”表示在命令提示符中显示当前工作目录的绝对路径。那么，如何表示当前时间呢？

Tom 决定从 Bash 的手册查找：

```
tom@geecy:~$ man bash↙
```

但 Bash 的手册共有五千多行，因此输入“ /PS1 ”向下搜索关于 PS1 的内容，然后不断输入“ / ”继续搜索。根据其中一个信息：

```
       PS1    The  value  of  this  parameter  is  expanded  (see
              PROMPTING  below)  and  used  as the primary prompt
              string.  The default value is ``\s-\v\$ ''.
```

提示“ see PROMPTING below ”。于是输入“ /PROMPTING ”重新搜索，最后找到关于时间显示的帮助：

```
              \t     the current time in 24-hour HH:MM:SS format
              \T     the current time in 12-hour HH:MM:SS format
              \@     the current time in 12-hour am/pm format
              \A     the current time in 24-hour HH:MM format
```

“ \@ ”正是 Tom 所需要寻找的。

Tom 执行命令：

```
tom@geecy:~$ PS1="\@ \w >"↙
02:06pm ~ > cd /usr/bin↙
02:06pm /usr/bin >
```

7.3 使用数值运算

【问题的提出】

Geecy 软件开发公司的开发人员 Tom 已经开发了如下的一个脚本 phone.sh ：

```
tom@geecy:~$ cat  phone.sh↙
#!/bin/bash
echo  -n  "Enter your name: "
read  Name
echo  -n  "Enter your phone number: "
read  Phone
echo  "$Name:$Phone"  >>  file.txt
```

该脚本将从键盘接收用户输入的名字和电话号码，然后以冒号分隔的形式添加一行到文件 file.txt 的末尾。

现在，Tom 需要修改该脚本的数据存放格式，在原格式的基础上增加一个编号，即以“ 编号:姓名:电话号码 ” 的格式保存每一行数据。要求编号从 1001 开始，以后每一行递增 1。

【问题分析】

Tom 已经有了解决思路：

(1) 首先使用一个辅助文件 count_file.txt 保存一个起始编号 1001；

(2) 从文件 count_file.txt 读入该编号保存到变量 Count；

(3) 从屏幕读取用户输入的姓名并保存到变量 Name；

(4) 从屏幕读取用户输入的电话号码并保存到变量 Phone；

(5) 以 “ 编号:姓名:电话号码 ” 的格式添加一行信息到数据文件 file.txt；

(6) 把变量 Count 的值增加 1；

(7) 把变量保存到文件 count_file.txt 并覆盖原来的内容；

使用这种方法，Tom 只需要对文件 count_file.txt 的内容手工做一次初始化，以后就可以重复运行该修改后的脚本 phone.sh，把用户输入的信息逐行添加到数据文件 file.txt。

但问题是，Shell 脚本中的变量都被当作是字符串。那么应该如何处理第 6 步的数值运算操作 “ 把变量 Count 的值增加 1 ”。

7.3.1 使用 expr 命令

expr 命令是一个简易的命令行计算器,能够进行一些简单的表达式(Expression)的计算。

例如：

```
tom@geecy:~$ expr 123 + 345↙
468
```

expr 命令需要一个表达式作为参数运行。expr 命令所使用的表达式除了支持具体的数值，还能够支持变量。例如：

```
tom@geecy:~$ x=12↙
tom@geecy:~$ expr $x + 34↙
46
```

当然，如果要计算的表达式只是单独一个变量，例如：

```
tom@geecy:~$ y=12↙
tom@geecy:~$ expr $y↙
12
tom@geecy:~$ echo $y↙
12
```

那么，在这种情况下其功能效果就等同于 echo 命令了。因此，expr 命令既可以把参数中的变量理解为数值变量，也可以把参数中的变量理解为字符串变量。Expr 命令判断的依据是：如果表达式中的运算符号(诸如“+”、“-”等)的左右两边都包含有空格，那么该运算符的两边的变量就被认为是数值变量，否则，就被认为是字符串变量。

所以，使用 expr 命令对表达式求值，必须注意保证每个运算符号的左右两边都留有空格。例如：

```
tom@geecy:~$ x=12↙
tom@geecy:~$ expr $x+34↙
12+34
tom@geecy:~$ echo $x+34↙
12+34
```

该例子中，变量 x 被认为是字符串变量。其效果等同于使用 echo 命令。

由于符号“*”已经被 Shell 用作通配符使用，因此如果进行乘法运算就需要使用反斜杠“\”对符号“*”转义，取消其特殊功能，恢复为普通的星号（乘号）。例如：

```
tom@geecy:~$ a=13↙
tom@geecy:~$ b=5↙
tom@geecy:~$ expr $a \* $b + 12 / 3↙
69
```

7.3.2 获取命令返回的结果

expr 命令的计算结果默认输出到屏幕。在 Shell 脚本中经常需要把计算结果先保存在某个变量，留待以后使用。

可以使用重定向功能实现这种功能：

```
tom@geecy:~$ expr $a \* $b + 12 / 3 > temp.txt↙
tom@geecy:~$ read Count < temp.txt↙
tom@geecy:~$ echo  $Count↙
69
```

这样的实现方法显然是比较繁琐的。Shell 提供了一个方便的特性，可以简化这类型的操作。通过使用一对反引号（``）把一个命令括起来，把该命令的屏幕输出结果截获并当作一个字符串来使用。该反引号在键盘上与波浪线符号（~）共用一个键。

例如：

```
tom@geecy:~$ date↙
Wed Jan 26 20:49:24 CST 2005
tom@geecy:~$ echo "Time is now `date`"↙
Time is now Wed Jan 26 20:49:25 CST 2005
```

命令 date 的输出结果被一对反引号截获并当作一个字符串（“Wed Jan 26 20:49:25 CST 2005”）来使用。

再例如：

```
tom@geecy:~$ whoami↙
tom
tom@geecy:~$ echo  "I am the user `whoami`"↙
I am the user tom
```

用户可以把这一特性应用于 expr 命令，以实现把计算结果保存在变量中的功能。因此，前面使用重定向功能实现的例子现在可以简化为：

```
tom@geecy:~$ Count=`expr $a \* $b + 12 / 3`↙
tom@geecy:~$ echo  $Count↙
69
```

7.3.3 使用算术展开

expr 命令是一个简易的命令行计算器，只能够进行一些简单表达式的计算。稍微复杂一点的表达式，例如是带有括号的表达式，expr 命令就无能为力了。

如果需要更灵活的运算，用户可以使用“$((…))”对表达式进行算术展开，即把里面的表达式当作是通常意义上的数学表达式进行数学运算，而不是当作一个普通的字符串。

例如：

```
tom@geecy:~$ echo $((123+345))↙
468
```

用户只需要简单地把一个普通的数学表达式放到里面，而且也没有那些空格之类的约束。算术展开所计算的结果默认只是一个字符串，用户需要使用 echo 命令才能输出到屏幕显示。

用户也可以选择把该计算结果保存到一个变量。例如：

```
tom@geecy:~$ p=$((123+345))↙
tom@geecy:~$ echo  $p↙
468
```

算术展开同样也支持含有变量的表达式。例如：

```
tom@geecy:~$ x=12↙
tom@geecy:~$ echo $(( $x+34 ))↙
46
```

不管是使用具体数值参加运算还是使用变量参加运算，运算符号的两边都不强制要求保留空格。但是适当地使用空格将大大提高命令或脚本的可读性。为了更好地提高可读性，算术展开所处理表达式中的变量还可以不需要使用符号“$”引用。例如：

```
tom@geecy:~$ x=12↙
tom@geecy:~$ echo $(( x+34 ))↙
46
```

使用算术展开，乘号不再需要使用转义方式表示。算术展开还可以支持多重括号的运算。例如：

```
tom@geecy:~$ x=13↙
tom@geecy:~$ echo $(( ((x+1)*2+4)/8 ))↙
4
```

该表达式含有两重括号。

【解决方案】

经过权衡比较，Tom 打算使用算术展开来处理第 6 个步骤的数值运算操作“把变量 Count 的值增加 1”。即：

```
Count=$(( Count + 1 ))
```

因此，Tom 使用 vi 把原来的脚本 phone.sh 修改为：

```
#!/bin/bash
read Count < count_file.txt
echo "[${Count}]:"
echo  -n  "Enter your name: "
read  Name
echo  -n  "Enter your phone number: "
read  Phone
echo  "$Count:$Name:$Phone"  >>  file.txt
Count=$(( Count + 1 ))
echo $Count > count_file.txt
```

运行该脚本之前，还需要对辅助文件“count_file.txt”进行初始化：

```
tom@geecy:~$ echo "1001" > count_file.txt↙
tom@geecy:~$ cat count_file.txt↙
1001
```

以后，就可以随时运行该修改后的脚本 phone.sh 增加一行信息到数据文件 file.txt：

```
tom@geecy:~$ ./phone.sh↙
[1001]:
Enter your name: ray↙
Enter your phone number: 123321↙
tom@geecy:~$ ./phone.sh↙
[1002]:
Enter your name: tom↙
Enter your phone number: 332211↙
```

可以使用 cat 命令检验结果：

```
tom@geecy:~$ cat file.txt↙
1001:ray:123321
1002:tom:332211
```

每运行一次脚本 phone.sh，只可以增加一行信息。如果信息比较多，那么操作起来还是不够方便。最后，Tom 采用了脚本的递归调用这种方式来简化操作，实现只需运行一次脚本 phone.sh 就可以连续增加若干行的信息：

```
#!/bin/bash
read Count < count_file.txt
echo "[${Count}]:"
echo  -n  "Enter your name: "
read  Name
echo  -n  "Enter your phone number: "
read  Phone
echo  "$Count:$Name:$Phone"  >>  file.txt
Count=$(( Count + 1 ))
echo $Count > count_file.txt
./phone.sh
```

由于脚本已经设置了执行权限，因此在执行方式上跟系统的命令程序是一样的。所以可以在脚本“ phone.sh ”的最后添加一行调用自身脚本的命令“ ./phone.sh ”。经过这样处理，其执行过程将得到很到程度的简化：

```
tom@geecy:~$ ./phone.sh↙
[1003]:
Enter your name: liluo↙
Enter your phone number: 12341234↙
[1004]:
Enter your name: huangdf↙
Enter your phone number: 87654321↙
```

如果不再需要增加信息，可以使用组合键<Ctrl>+c 强行终止脚本的执行。

7.4 控制脚本流程

【问题的提出】

Geecy软件开发公司的开发人员Tom所开发的通讯录管理脚本phone.sh的改进版本如下：

```
tom@geecy:~$ cat  phone.sh↙
#!/bin/bash
read Count < count_file.txt
echo "[${Count}]:"
echo  -n  "Enter your name: "
read  Name
echo  -n  "Enter your phone number: "
read  Phone
echo  "$Count:$Name:$Phone"  >>  file.txt
Count=$(( Count + 1 ))
echo $Count > count_file.txt
./phone.sh
```

该版本比旧版本已经有了很大的改进，能够满足一般的使用。但是根据 Tom 的开发经验，该版本至少存在四大缺陷：

(1) 脚本运行的稳定性不高，因为该脚本严重依赖于辅助文件 count_file.txt，一旦该文件被以外破坏，该脚本将不能正常工作；

(2) 运行性能不高，因为该脚本在运行过程中需要频繁读写三个文件，而从该脚本的功能来看，运行过程中需要频繁读写的文件应该只有数据文件“ file.txt ”；

(3) 功能单一，因为目前只有增加功能，还缺乏基本的搜索功能；

(4) 用户界面不够友好，应该使用菜单式的用户界面。

因此 Tom 需要继续改进该脚本。

【问题分析】

如果要提高稳定性，就要想办法避免使用辅助文件 count_file.txt。使用该辅助文件的原意其实就是用来保存数据文件 file.txt 中的当前最大编号的值，以便在增加一行数据到数据文件的时候可以凭此得到下一个编号的值。事实上，另外保存该最大值纯属多余，因为该最大值就在数据文件 file.txt 的最后一行的第一个数据域。只需要想办法在增加一行新数据之前把该值读出来，然后增加一，就可以构造出下一个编号。

Linux 系统中有一个适合使用的命令 tail，能够从一个文本文件的最后一行起，连续抽取若干行数据显示到屏幕。

tail 命令的简单使用如下：

```
tom@geecy:~$ tail  /etc/passwd↙
```

```
nobody:x:65534:65534:nobody:/home:/bin/sh
ray:x:1000:1000:Debian User,,,:/home/ray:/bin/bash
identd:x:100:65534::/var/run/identd:/bin/false
sshd:x:101:65534::/var/run/sshd:/bin/false
tom:x:1001:100::/home/tom:/bin/bash
hdf:x:2046:1000:friend:/tmp/hdf:/bin/tcsh
David:x:3001:2001::/home/David:/bin/bash
Peter:x:3002:2001::/home/Peter:/bin/bash
Jack:x:3003:2002::/home/Jack:/bin/bash
Mike:x:3004:2002::/home/Mike:/bin/bash
```

这个例子中使用文件“/etc/passwd”作参数，tail 命令从该文件的最后一行起，连续抽取默认值为 10 行的内容显示到屏幕上。

命令 tail 提供了一个选项“-n”供用户指定需要显示的行数。例如：

```
tom@geecy:~$ tail  -n 1 /etc/passwd↙
Mike:x:3004:2002::/home/Mike:/bin/bash
```

选项“-n 1”指定从该文件的最后一行起，连续抽取 1 行的内容显示到屏幕上，也就是只显示文件的最后一行。

Tom 可以利用 tail 命令与 cut 命令的组合来取代辅助文件的使用。

如果上述的稳定性问题解决了，那么只需要避免使用脚本的递归调用就可以实现整个运行过程只需要频繁读写一个文件，即数据文件 file.txt。使用脚本的递归调用，目的其实就是希望能够重复运行某一部分脚本。只要找到能够重复运行某一部分脚本的替代方法，那么脚本的递归调用这种方法就可以“退役”了。

对于搜索功能的实现，可以使用 grep 命令来完成。

而菜单式的用户界面的实现原理，就是首先打印若干个菜单项的内容到屏幕上，等待用户选择其中一个功能，脚本程序根据用户的键盘输入来判断用户的选择，从而进行相应功能的操作。操作任务完成后，再次打印若干个菜单项的内容到屏幕上，如此周而复始。

7.4.1 测试表达式

test 命令能够对表达式进行测试并求出 True 或 False 的逻辑值。如果该表达式符合客观事实，那么测试的结果是真（True），如果该表达式符合客观事实，那么测试的结果是假（False）。

脚本程序可以根据这些测试结果的真与假的不同逻辑值作出判断，以表现出不同的行为。

test 命令可以对字符串表达式和算术表达式测试，也可以对文件（或目录）进行测试。例如，比较简单的测试是对一个比较两个字符串是否相等的字符串表达式进行测试：

```
test  $Name = "Linus"
```

如果变量$Name 的内容刚好是"Linus"，那么此测试的结果的逻辑值为真。如果变量$Name 的内容是"stallman"，那么此测试的结果的逻辑值为假。用户给变量$Name 赋予不同内容的

值就可以有两个不同的测试结果。也就是说，用户可以通过控制变量$Name 的内容来实现控制脚本程序的下一步该执行什么操作。

test 命令的语法要求被测试的表达式中的比较运算符（例如这里的“ = ”）的左右两边必须有空格。这一点与 expr 命令相似。

test 命令可以一次测试多个表达式。这些表达式之间需要使用选项“ -a ”或“ -o ”连接。选项“ -a ”表示“ 与 ”(AND)运算，选项“ -o ”表示“ 或 ” (OR)运算。

例如：

```
test  $Name = "Linus"  -a  $Country = "China"
```

test 命令使用选项“ -a ”连接“ $Name = "Linus" ”和“ $Country = "China" ”。test 命令首先分别求出这两个表达式的逻辑值，再把这两个值进行“ 与 ”运算。假如表达式“ $Name = "Linus" ”的逻辑值为真，而表达式“ $Country = "China" ”的逻辑值为假，那么“ 真 ”与“ 假 ”这两个逻辑值进行“ 与 ”运算的结果就为假。

“ 与 ”运算的规则是只有参加运算的两个值都为真，那么结果才为真；其他情况都为假。“ 或 ”运算的规则是只有参加运算的两个值都为假，那么结果才为假；其他情况都为真。

为了有更好的可读性，用户可以使用方括号“ [ ] ”来代替 test 命令。例如，上述的两个例子可以分别修改为：

```
[  $Name = "Linus"  ]
```

和

```
[  $Name = "Linus"  -a  $Country = "China"  ]
```

除了比较运算符的两边必须有空格之外，这里还要求“ [”的后面和“] ”的前面也必须有空格。

7.4.2 设计分支结构

脚本程序中的命令默认是顺序执行的。也就是说，一个命令执行完毕，下一个命令就立刻无条件地开始执行。分支结构则提供给用户一种流程控制的能力，能够让用户根据某些条件作出判断，然后有所选择地执行某些语句。

- **使用 if 语句**

用户可以使用 if 语句来实现分支结构。

if 语句的基本格式为：

```
if  <Condition>
then
    <Commands>
fi
```

其中：

（1）<Condition>是一个条件表达式，其逻辑值要么为真，要么为假；

（2）<Commands>表示若干个命令，fi 是 if 语句的结束标志；

（3）如果<Condition>的逻辑值为真，那么就逐一执行<Commands>所表示的若干个命令，然后结束 if，继续执行 fi 后面的其他语句。

（4）如果<Condition>的逻辑值为假，那么就直接结束 if 语句，然后执行 fi 后面的语句。<Commands>所表示的若干个命令不执行。

例如：

```
tom@geecy:~$ cat  test.sh↙
#!/bin/bash
echo  -n  "Do you know the answer? (y/n) "
read  Result
if  [  $Result  =  "y"  ]
then
    echo  "Good!"
    echo  "Congratulations!"
fi
echo  "Finished!"
```

该脚本首先提示用户输入一个字符“y”或者“n”，然后从键盘接收用户的输入并保存到变量 Result 中。接着 if 语句根据其后的表达式的测试值进行判断，如果该值为真（即用户输入的字符是“y”），那么就执行 then 与 fi 之间的两个 echo 命令，然后继续执行 fi 后面的 echo 语句；如果该值为假（即用户输入的字符不是“y”），那么就直接执行 fi 后面的 echo 语句。即脚本的运行过程如下：

```
tom@geecy:~$ ./test.sh↙
Do you know the answer? (y/n) y↙
Good!
Congratulations!
Finished
tom@geecy:~$ ./test.sh↙
Do you know the answer? (y/n) n↙
Finished
```

if 语句的另一种格式为：

```
if  <Condition>
then
    <Commands1>
else
    <Commands2>
fi
```

其中：

（1）<Condition>是一个条件表达式，其逻辑值要么为真，要么为假；

（2）<Commands1>和<Commands2>分别表示若干个命令；

（3） 如果<Condition>的逻辑值为真，那么就逐一执行<Commands1>所表示的若干个命令，然后结束 if，继续执行 fi 后面的其他语句。

（4）如果<Condition>的逻辑值为假，那么就逐一执行<Commands2>所表示的若干个命令，然后结束 if，继续执行 fi 后面的其他语句。

例如：

```
tom@geecy:~$ cat  test.sh↙
#!/bin/bash
echo  -n  "Do you know the answer? (y/n) "
read  Result
if  [  $Result  =  "y"  ]
then
    echo  "Good!"
    echo  "Congratulations!"
else
    echo  "Bad!"
    echo  "Work hard!"
fi
echo  "Finished!"
```

该脚本首先提示用户输入一个字符“y”或者“n”，然后从键盘接收用户的输入并保存到变量 Result 中。接着 if 语句根据其后的表达式的测试值进行判断，如果该值为真（即用户输入的字符是“y”），那么就执行 then 与 else 之间的两个 echo 命令，然后继续执行 fi 后面的 echo 语句；如果该值为假（即用户输入的字符不是“y”），那么就执行 else 与 fi 之间的两个 echo 命令，然后继续执行 fi 后面的 echo 语句。即脚本的运行过程如下：

```
tom@geecy:~$ ./test.sh↙
Do you know the answer? (y/n) y↙
Good!
Congratulations!
Finished
tom@geecy:~$ ./test.sh↙
Do you know the answer? (y/n) n↙
Bad!
Work hard!
Finished
```

if 语句的完整格式可以描述为：

```
if  <Condition1>
then
    <Commands1>
[elif <Condition2>
then
    <Commands2>]
[elif <Condition3>
then
    <Commands3>]
... ...
[else
    <Commands>]
fi
```

其中：

（1）中括号部分表示是可选的；

（2）elif 的意义是“else if”。elif 可以连续出现若干次或者零次；

（3）else 最多只能出现一次。

（4）Shell 依次检查 if 和 elif 后面的各个条件表达式，当某个条件为真时就执行后面相应的若干个命令，然后结束 if 语句。

（5）当 if 和 elif 后面的所有条件表达式都为假时执行 else 后面的若干个命令，然后结束 if 语句。

有了 elif，用户可以写出更复杂的分支结构。例如下面是一个判断用户输入的字母是否为元音字母的脚本：

```
tom@geecy:~$ cat  vowel.sh↙
#!/bin/bash
echo   -n  "Please input a vowel letter (Aa Ee Ii Oo Uu): "
read   Key
if  [  $Key  =  "A"  -o  $Key  =  "a"  ]
then
    echo  "You input the vowel letter \"A\"."
     echo   "Thank you."
elif  [  $Key  =  "E"  -o  $Key  =  "e"  ]
then
    echo  "You input the vowel letter \"E\"."
     echo   "Thank you."
elif  [  $Key  =  "I"  -o  $Key  =  "i"  ]
then
    echo  "You input the vowel letter \"I\"."
```

```
    echo  "Thank you."
elif [ $Key = "O" -o $Key = "o" ]
then
    echo  "You input the vowel letter \"O\"."
    echo  "Thank you."
elif [ $Key = "U" -o $Key = "u" ]
then
    echo  "You input the vowel letter \"U\"."
    echo  "Thank you."
else
    echo  "The letter you input is not a vowel letter."
    echo  "Try again."
fi
echo "Finished."
```

脚本运行后首先提示用户输入其中一个元音字母的字符，然后从键盘接收用户输入的字符并保存到变量 Key，然后依次检查 if 和 elif 后面的各个条件表达式，如果有逻辑值为真的条件表达式（即用户输入的字母是一个元音字母），那么就执行相应的条件表达式后的两个 echo 命令，否则，就执行 else 后的两个 echo 命令。

脚本使用了选项“ -o ”连接两个表达式进行“ 或 ”的逻辑运算。为了在 echo 命令输出一个双引号，脚本使用了反斜杠“ \ ”作转义。

■ **使用 case 语句**

当 if 语句的出口分支比较多的时候，如果适合使用 case 语句实现，那么可以提高脚本的可读性。

case 语句的使用格式是：

```
case <Variable> in
    Pattern1)
      <Commands1>
      ;;
    Pattern2)
      <Commands2>
      ;;
    ... ...
    *)
      <Commands>
      ;;
esac
```

其中：

（1） case 语句的执行过程是把变量 Variable 的内容依次与模式 Pattern1、Pattern2 等匹配，一旦匹配成功就执行其后的若干个命令，然后结束 case 语句。

（2） 模式 Pattern1、Pattern2 等采用的规则与文件通配符相同。

（3） 如果前面的所有模式都没有匹配，则匹配最后的“ * ”通配符。

（4） esac 是 case 语句的结束标志。

例如：

```
#!/bin/bash
echo    -n   "Enter  a  word    (dozen or score) : "
read    Str
case    $Str    in
     dozen)
          echo    "12"
          ;;
     score)
          echo    "20"
          ;;
     *)
        echo   "It  is  neither  a  dozen  nor  a  score."
          ;;
esac
```

本例中，如果用户输入字符串“ dozen ”则输出“ 12 ”，如果用户输入字符串“ score ”则输出“ 20 ”。如果输入的字符串都不是这两个字符串之一，则匹配通配符“ * ”并输出相应的提示信息“ It is neither a dozen nor a score. ”。

对于前面给出的使用 if 语句的判断元音字母的脚本，用户可以使用 case 语句把它改写为：

```
#!/bin/bash
echo    -n   "Please input a vowel letter: (Aa Ee Ii Oo Uu)"
read    Key
case    $Key    in
     [Aa])
        echo   "You  input  the  vowel  letter  \"A\"."
          echo    "Thank  you."
          ;;
     [Ee])
        echo   "You  input  the  vowel  letter  \"E\"."
          echo    "Thank  you."
          ;;
     [Ii])
```

```
        echo   "You input the vowel letter \"I\"."
         echo    "Thank you."
         ;;
    [Oo])
        echo   "You input the vowel letter \"O\"."
         echo    "Thank you."
         ;;
    [Uu])
        echo   "You input the vowel letter \"U\"."
         echo    "Thank you."
         ;;
    *)
       echo   "The letter you input is not a vowel letter."
         echo    "Try again."
         ;;
esac
echo  "Finished."
```

由于 case 语句可以使用通配符，因此该脚本使用通配符“ [ ] ”轻松实现大小写的同时匹配。

7.4.3 测试字符串

对字符串的测试，除了测试是否相等，还有其他的一些常用的测试。表 7-1 列出了常用的字符串测试选项。

表 7-1

选　项	作　用
string	如果 string 的长度不为零，则值为 True，否则为 False
-z string	如果 string 的长度为零，则值为 True，否则为 False
String1 = String2	如果 string1 与 string2 相等，则值为 True，否则为 False
String1 != String2	如果 string1 与 string2 不相等，则值为 True，否则为 False

例如：

```
#!/bin/bash
echo    -n   "Please input a string: "
read    Str1
if    [    -z    $Str1     ]
then
      echo     "The string can not be null."
      echo     "Try again."
       exit
```

```
fi
echo    -n   "Please  input  another string: "
read    Str2
if    [    $Str2    ]
then
      echo     "The length of the string is not equal to zero."
else
      echo     "The string can not be null."
      echo     "Try again."
       exit
fi
if    [    $Str1    =    $Str2    ]
then
     echo   "The  two  strings  are  equal."
Fi
if    [    $Str1    !=    $Str2    ]
then
     echo   "The  two  strings  are  not  equal."
fi
echo  "Finished."
```

本例子演示了表 7-1 中的四种字符串测试选项的用法。

脚本从键盘接收用户输入的一个字符串，然后检查该字符串是否为空（即长度是否为零），若为空，则输出出错信息，并用 exit 命令退出运行该脚本的子 Shell。

若该字符串不为空，则再提示用户输入另一个字符串，然后检查该字符串的长度是否不为零，若长度不为零，则给出“ The length of the string is not equal to zero. ”的信息，若长度为零，则给出出错信息，并用 exit 命令退出运行该脚本的子 Shell。

如果两个字符串都非空，则比较两个字符串是否相等，并分别给出相应的提示信息。

7.4.4 测试算术式

由于 Shell 的变量是字符串类型，因此如果要把变量的内容作为数值进行比较，则需要使用专门的算术测试选项。表 7-2 列出了六种算术测试选项。

表 7-2

选　项	作　用
-eq	等于(Equal)
-ne	不等于(Not Equal)
-gt	大于(Greater Then)
-ge	大于(Greater Then)或等于(Equal)

-lt	小于(Less Then)
-le	小于(Less Then)或等于(Equal)

例如：

```
#!/bin/bash
echo   -n  "Please input number 1: "
read   Num1
echo   -n  "Please input number 2: "
read   Num2
echo   -n  "Please input number 3: "
read   Num3
Max=$Num1
if  [   $Num2   -gt   $Max   ]
then
      Max=$Num2
fi
if  [   $Num3   -gt   $Max   ]
then
      Max=$Num3
fi
echo  "The maximum number is "$Max
```

本例子使用两个“大于”（-gt）比较实现求三个数中的最大数的功能

7.4.5 测试文件

除了测试字符串和算术式，用户还经常需要测试文件（或目录）的状态。表 7-3 列出了文件测试的常用选项。

表 7-3

选　项	作　用
-e File	如果文件 File 存在(Exist)，则为 True
-s File	如果文件 File 存在且文件大小(Size)大于零，则为 True
-f File	如果文件 File 存在且是普通文件(File)，则为 True
-d File	如果文件 File 存在且是目录(Directory)，则为 True
-b File	如果文件 File 存在且是块(Block)特殊文件，则为 True
-c File	如果文件 File 存在且是字符(Character)特殊文件，则为 True
-L File	如果文件 File 存在且是符号链接(Link)文件，则为 True
-r File	如果文件 File 存在且是可读的(Readable)，则为 True
-w File	如果文件 File 存在且是可写的(Writable)，则为 True
-x File	如果文件 File 存在且是可执行的(Executable)，则为 True

-O File	如果文件 File 存在且属于当前用户(Owner)，则为 True
-G File	如果文件 File 存在且属于当前用户组(Group)，则为 True
File1 -nt File2	如果文件 File1 新于(Newer Then) File2，则为 True
File1 -ot File2	如果文件 File1 旧于(Older Then) File2，则为 True

例如：

```
#!/bin/bash
echo    -n  "Please input the file name: "
read    FileName
if   [   !   -e   $FileName   ]
then
     echo   "File "${FileName}" not exist."
      exit
fi
if   [   -f   $FileName   ]
then
     echo   "File "${FileName}" is an ordinary file."
elif   [   -d   $FileName   ]
then
     echo   "File "${FileName}" is a directory file."
else
    echo   "File "${FileName}" is neither an ordinary file nor a directory file."
fi
```

本例中，脚本接收用户输入的文件名后，判断该文件是否存在。如果不存在，则输出出错信息并结束。

如果存在，则判断是普通文件还是目录文件，输出相应的信息。如果既不是普通文件也不是目录文件，则执行 else 后面的语句输出相应的信息。

符号“！”表示把逻辑值取反，即如果原来的值是真，则变为假，原来的值是假，则变为真。

7.4.6 设计循环结构

循环结构提供了重复执行一段脚本的能力。脚本程序可以使用两种风格的循环语句来实现循环结构。

■ while 循环

使用 while 语句的格式是：

```
while  <Condition>
do
     <Commands>
done
```

其中：

（1） 当条件 Condition 为真时执行 do 和 done 之间的命令 Commands。

（2） 当命令 Commands 执行完成后，while 语句立即再次测试条件 Conditon，如果结果为真，则继续执行循环体 Commands，否则结束 while 循环。

例如：

```
#!/bin/bash
Count=1
while   [   $Count   -le   10   ]
do
      echo   $Count
    Count=$(( Count + 1 ))
done
```

本例中，将输出从 1 到 10 的整数。这是 while 语句用于固定循环次数的一个例子。while 语句更适合于不确定循环次数的循环。例如：

```
#!/bin/bash
isExit=n
while   [   $isExit   =   "n"   ]
do
      echo   -n  "Please input a file name: "
      read   FileName
      cat   $FileName
      echo
      echo   -n  "Do you want to exit? (y/n) "
      read   isExit
done
```

脚本程序首先提示用户输入一个文件名，接着显示该文件的内容，然后询问用户是否退出。用户输入“n”继续，输入其他任何字符将退出本次的执行。在这里，循环的次数是在运行的过程中由用户决定。

除了使用在固定循环次数和不确定循环次数的两种情况外，while 语句还可以使用于无限循环。

例如：

```
#!/bin/bash
while true
do
    echo   "I like Linux."
done
```

执行上面的脚本将导致屏幕上输出大量的“I like Linux.”。用户可以通过使用组合键 <Ctrl> + c 强行终止该脚本的执行。“true”表示条件为真。

■ **for 循环**

for 语句实现循环结构的一种格式是：

```
for(( Expression1; Expression2; Expression3 ))
do
    <Commands>
done
```

其中：

(1) for 语句首先执行 Expression1，这是一个初始化的表达式。

(2) 然后检查 Expression2 的逻辑值，如果值为真，则执行循环体的命令 Commands，如果值为假，则结束 for 语句。

(3) 每当执行完循环体 Commands 后，for 语句立即执行 Expression3，然后检查 Expression2 的逻辑值，如果值为真，则执行循环体的命令 Commands，如果值为假，则结束 for 语句。

(4) 总而言之，每次执行循环体的命令 Commands 之前，总是先检查 Expression2 的逻辑值。

例如：

```
#!/bin/bash
for(( Count=1; Count<=10; Count=Count+1 ))
do
    echo  $Count
done
```

本例完成前面一个例子的同样功能。

除了固定次数的循环，for 语句也能用于不确定次数的循环。例如，可以把前面的例子改写如下：

```
#!/bin/bash
for((  ; ;  ))
do
    echo  -n  "Please input a file name: "
    read  FileName
    cat  $FileName
    echo
    echo  -n  "Do you want to exit? (y/n) "
    read  isExit
    if  [  $isExit  !=  "n"  ]
    then
        exit
    fi
done
```

for 语句的三个表达式都省略，表示是无限次循环。本例中使用了 exit 命令退出整个脚本程序。

for 语句实现循环结构的另一种格式是：

```
for  <Variable>  in  <List>
do
    <Commands>
done
```

其中：

(1) 列表 List 的各个值将依次赋给变量 Variable。对于每一次的赋值，都执行循环体 Commands 一次。

(2) 若列表 List 有 n 个值，则 for 语句循环执行 n 次。

例如：

```
#!/bin/bash
for  Name  in  "Jack"  "C++ Programming"  "Linus Torvalds"
do
    echo  $Name
done
```

本例中，in 后面的三个值依次赋给变量 Name，并循环执行三次命令“echo $Name”。输出结果为：

```
Jack
C++ Programming
Linus Torvalds
```

这种格式的 for 语句比较适合于文件和目录的操作。例如：

```
#!/bin/bash
for  FileName  in  `ls  ~`
do
    if  [  -f  ~/$FileName  ]
    then
        echo  "The contents of ~/${FileName} is: "
          cat  ~/$FileName
    fi
done
```

命令“ls ~”列出用户起始目录的所有文件名和目录名，并依次赋给变量 FileName，循环执行 do 与 done 之间的命令。循环体的功能是首先对该文件进行测试，如果是普通文件则用 cat 命令显示其内容到屏幕。

【解决方案】

（1）初始化编号

脚本程序从数据文件读取当前的最大编号，然后增加 1，可以使用组合命令

```
Count=`tail -n 1 file.txt | cut -d ":" -f1`
```

来完成。但文件 file.txt 不一定存在，因此使用选项“ -s ”测试文件，如果文件存在且文件大小是大于 0，才使用上述的组合命令读取当前的最大编号，然后增加 1。否则，需要使用：

```
Count="1001"
```

创建变量并赋值 1001，作为初始化的编号。

考虑到经常需要引用数据文件的名称 file.txt，因此定义一个变量 File 保存该名称，以方便随时修改该文件名。

因此，综合如下：

```
#!/bin/bash
File="file.txt"
if [ -s $File ]
then
    Count=`tail -n 1 $File | cut -d ":" -f1`
    Count=$(( Count + 1 ))
else
    Count="1001"
fi
```

(2) 编写一个菜单式的用户界面：

clear 命令用于清除屏幕。使用 while 语句实现无限次循环。当然，遇到 exit 命令可以结束循环。用户菜单的选择，使用 case 语句实现，变量 Key 用于保存用户的键盘输入。如果输入 0 则退出脚本程序。

```
while true
do
    clear
    echo "[0]. Exit"
    echo "[1]. Add phone number"
    echo "[2]. Search name or phone number"
    echo -n "Your choice (0,1 or 2): "
    read Key
    case $Key in
      0)
          exit;;
      1)  ;;
      2)  ;;
    esac
done
```

（3）原来已经实现的增加数据的功能。可以直接放在 case 语句的“ 1）”选项后。

```
        clear
        echo "[${Count}]:"
        echo  -n  "Enter your name: "
        read  Name
        echo  -n  "Enter your phone number: "
        read  Phone
        echo  "$Count:$Name:$Phone"  >>  file.txt
        Count=$(( Count + 1 ))
```

（4）搜索功能。首先清屏，然后从键盘读入用户的搜索关键字并保存到变量 Word。使用 grep 命令实现搜索。

```
        clear
        echo -n "Input a keyword: "
        read Word
        clear
        grep $Word $File
        echo "Press enter to continue..."
        read Enter
```

最后，合并为一个完整脚本程序如下：

```
#!/bin/bash
File="file.txt"
if [ -s $File ]
then
    Count=`tail -n 1 file.txt | cut -d ":" -f1`
    Count=$(( Count + 1 ))
else
    Count="1001"
fi
while true
do
    clear
    echo "[0]. Exit"
    echo "[1]. Add phone number"
    echo "[2]. Search name or phone number"
    echo -n "Your choice (0,1 or 2): "
    read Key
    case $Key in
```

```
    0)
       exit;;
    1)
       clear
       echo "[${Count}]:"
       echo  -n  "Enter your name: "
       read  Name
       echo  -n  "Enter your phone number: "
       read  Phone
       echo  "$Count:$Name:$Phone"  >>  file.txt
       Count=$(( Count + 1 ))
       ;;
    2)
       clear
       echo -n "Input a keyword: "
       read Word
       clear
       grep $Word $File
       echo "Press enter to continue..."
       read Enter
       ;;
   esac
done
```

7.5 练 习

1、 Tenny 需要修改命令提示符，使得当前工作目录是“ /usr/bin ”的时候显示：

[bin]$

当前工作目录是“ /usr/local/sbin ”的时候显示：

[sbin]$

2、 编写一个 Shell 脚本，从键盘读入 5 个整数。然后显示最大数、最小数以及平均值。

3、 编写一个 Shell 脚本，显示 Fibonacci 数列的前 20 项。例如：

0，1，1，2，3，5，8，13，21...

4、 编写一个 Shell 脚本，从键盘上接收两个文件名，如果两个文件都存在则交换两个文件的内容，否则应给出出错信息。

5、 编写一个 shell 脚本，产生并输出如下的序列：

1，2，3，2，3，4，3，4，5，4，5，6… 98，99，100

6、 Cell 宽带数据公司的客户服务中心每天都会接收到一些客户电话。值班人员为客户解答完毕后，客户可以通过电话按键对该值班人员的工作作出“ 满意 ”或“ 不满意 ”的投票评价，如果直接挂机则表示放弃投票。

请编写一个 shell 脚本，读入每天电话的总数、“ 满意 ”的票数和“ 不满意 ” 的票数，统计放弃投票的数量。

【提示】 放弃投票的数量 = 总数量 –“ 满意 ”的票数–“ 不满意 ”的票数。

7、 Cell 宽带数据公司的客户服务中心对值班员工的工作质量进行跟踪评价，评价标准是客户对值班员工的“ 满意 ”投票数量与总投票数量的比值。如下表所示：

比 值(%)	等 级
Less then 70	Average
Between 70 and 95	Good
Greater then 95	Excellent

Tenny 需要编写一个 shell 脚本，读入“ 满意 ”投票数量和总投票数量，然后计算并输出相应的评定等级。

8、 Cell 宽带数据公司向客户提供一种菜单式的查询服务，如下表所示：

序 号	服 务 名 称	服 务 内 容
1	查询费用(Bill query)	显示客户每月的费用
2	业务咨询(Consulting)	获取业务的使用方法
3	最新消息(News)	显示产品推广和优惠活动等
4	投诉(Complain)	接受客户的投诉和建议

Tenny 需要编写一个 shell 脚本，显示服务菜单。当客户选择某一项服务后输出相应的服务内容。

9、 Cell 宽带数据公司需要录入客户的资料并保存到文件，每个客户的资料保存在单独的一行。格式如下：

客户号：姓名：地址：电话：邮政编码

其中要求客户号由系统自动产生，从 10001 起。

Tenny 需要编写一个 Shell 脚本，提示输入每项数据，并按照规定的格式保存到文件。录入完一个客户的资料后，提示是否继续。按“ y ”继续录入下一客户，“ n ”退出。

参 考 文 献

[1] David Tansley 著．张春萌等译．LINUX 与 UNIX Shell 编程指南．北京：机械工业出版社，2000 年

[2] Lamb,L.，Robbins，A．编著．莫蓉蓉等译．学习 vi 编辑器（第六版）．北京：机械工业出版社，2003 年

[3] 王波．FreeBSD 使用大全（第 2 版）．北京：机械工业出版社，2000 年

[4] 张尧学，史美林．计算机操作系统教程．北京：清华大学出版社，1993 年

[5] http://www.kernel.org

[6] http://www.linux.org

[7] http://www.gnu.org

[8] http://www.bell-labs.com/history/unix/

[9] http://www.cs.vu.nl/~ast/minix.html

[10] http://www.debian.org

[11] http://www.slackware.org

[12] http://www.mandrakelinux.com

[13] http://www.redhat.com

[14] http://www.suse.com

[15] http://www.turbolinux.com